ASM330LHB: machine learning core

Introduction

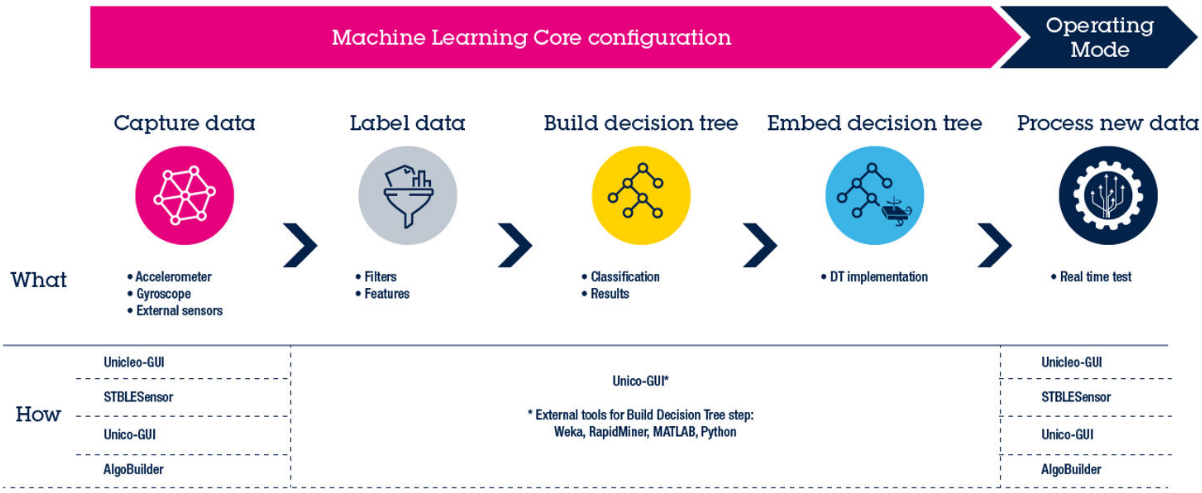
This document provides information on the machine learning core feature available in the ASM330LHB. The machine learning processing capability allows moving some algorithms from the application processor to the MEMS sensor, enabling consistent reduction of power consumption.

The machine learning processing capability is obtained through decision-tree logic. A decision tree is a mathematical tool composed of a series of configurable nodes. Each node is characterized by an "if-then-else" condition, where an input signal (represented by statistical parameters calculated from the sensor data) is evaluated against a threshold.

The ASM330LHB can be configured to run up to 8 decision trees simultaneously and independently. The decision trees are stored in the device and generate results in the dedicated output registers.

The results of the decision tree can be read from the application processor at any time. Furthermore, there is the possibility to generate an interrupt for every change in the result in the decision tree.

Figure 1. Machine learning core supervised approach



1 Machine learning core in the ASM330LHB

The machine learning core (together with the finite state machine) is one of the main embedded features available in the ASM330LHB. It is composed of a set of configurable parameters and decision trees able to implement algorithms in the sensor itself.

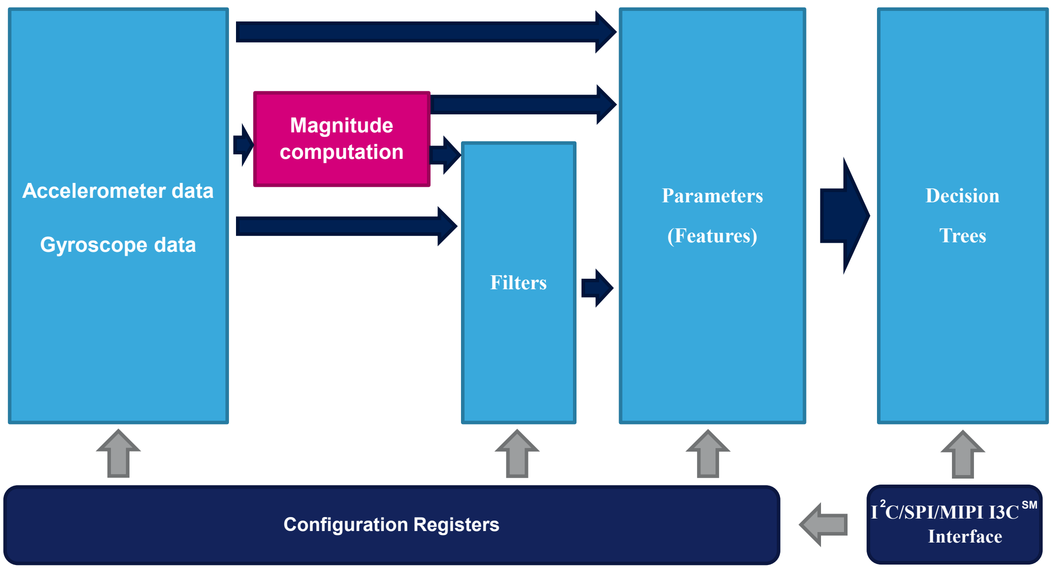
The kind of algorithms suitable for the machine learning core are those which can be implemented by following an inductive approach, which involves searching patterns from observations. Some examples of algorithms that follow this approach are: activity recognition, fitness activity recognition, motion intensity detection, vibration intensity detection, carrying position recognition, context awareness, false-positive rejection, and so forth.

The idea behind the machine learning core is to use the accelerometer and gyroscope sensor data to compute a set of statistical parameters selectable by the user (such as mean, variance, energy, peak, zero-crossing, and so forth) in a defined time window. In addition to the sensor input data, some new inputs can be defined by applying some configurable filters available in the device.

The machine learning core parameters are called “features” and can be used as input for a configurable decision tree that can be stored in the device.

The decision tree that can be stored in the ASM330LHB is a binary tree composed of a series of nodes. In each node, a statistical parameter (feature) is evaluated against a threshold to establish the evolution in the next node. When a leaf (one of the last nodes of the tree) is reached, the decision tree generates a result that is readable through a dedicated device register.

Figure 2. Machine learning core in the ASM330LHB



The machine learning core output data rate can be configured among one of the four available rates from 12.5 to 104 Hz. The bits MLC_ODR in the embedded function register EMB_FUNC_ODR_CFG_C (60h) allow selecting one of the four available rates as shown in the following table.

Table 1. Machine learning core output data rates

MLC_ODR bits in EMB_FUNC_ODR_CFG_C (60h)	Machine learning core output data rate
00	12.5 Hz
01	26 Hz (default)
10	52 Hz
11	104 Hz

In order to implement the machine learning processing capability of the ASM330LHB, it is necessary to use a "supervised learning" approach that consists of:

- Identifying some classes to be recognized
- Collecting multiple data logs for each class
- Performing some data analysis from the collected logs to learn a generic rule that allows mapping inputs (data logs) to outputs (classes to be recognized)

In an activity recognition algorithm, for instance, the classes to be recognized might be: stationary, walking, jogging, biking, driving, and so forth. Multiple data logs have to be acquired for every class, for example multiple people performing the same activity.

The analysis on the collected data logs has the purpose of:

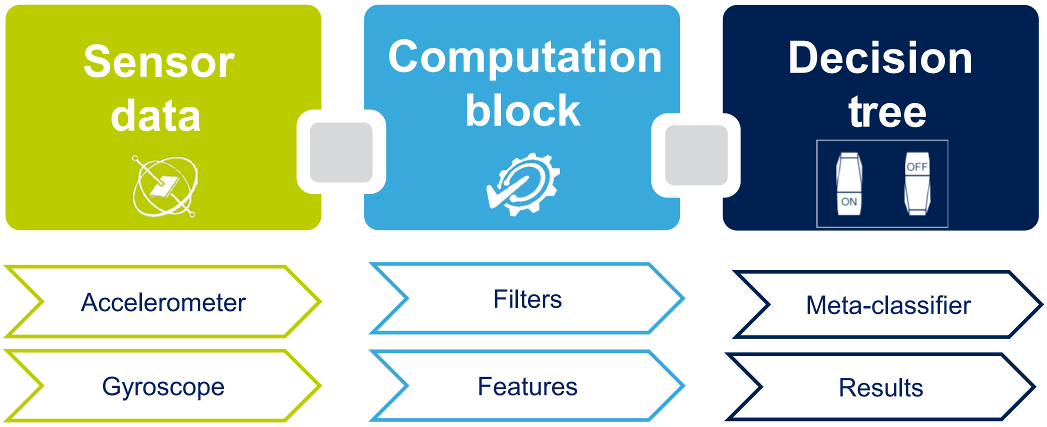
- Defining the features to be used to correctly classify the different classes
- Defining the filters to be applied to the input data to improve the performance using the selected features
- Generating a dedicated decision tree able to recognize one of the different classes (mapping inputs to outputs)

Once a decision tree has been defined, a configuration for the device can be generated by the software tool provided by STMicroelectronics (described in Section 2 Machine learning core tools). The decision tree runs on the device, minimizing the power consumption.

Going deeper in detail in the machine learning core feature inside the ASM330LHB, it can be thought of as three main blocks (Figure 3):

1. Sensor data
2. Computation block
3. Decision tree

Figure 3. Machine learning core blocks



The first block, called "Sensor data", is composed of data coming from the accelerometer and gyroscope, which are built in the device.

The machine learning core inputs defined in the first block are used in the second block, the "Computation block", where filters and features can be applied. The features are statistical parameters computed from the input data (or from the filtered data) in a defined time window, selectable by the user.

The features computed in the computation block are used as input for the third block of the machine learning core. This block, called "Decision tree", includes the binary tree that evaluates the statistical parameters computed from the input data. In the binary tree the statistical parameters are compared against certain thresholds to generate results (in the example of the activity recognition described above, the results were: stationary, walking, jogging, biking, and so forth). The decision tree results might also be filtered by an optional filter called "meta-classifier". The machine learning core results are the decision tree results, which include the optional meta-classifier.

The machine learning core memory is organized in a “dynamic” or “modular” way, in order to maximize the number of computation blocks that can be configured in the device (filters, features, and so forth). A dedicated tool has been designed to generate the configuration of the ASM330LHB, in order to automatically manage memory usage. The tool is available in the Unico GUI and it is described later in Section 2 Machine learning core tools.

The following sections explain in detail the three main blocks of the machine learning core in the ASM330LHB described in Figure 3.

1.1 Inputs

The ASM330LHB works as a combo (accelerometer + gyroscope) sensor, generating acceleration and angular rate output data. The 3-axis data of the acceleration and angular rate can be used as input for the machine learning core. Figure 4 and Figure 5 show the inputs of the machine learning core block in the accelerometer and gyroscope digital chains. The position of the machine learning core (MLC) block in the two digital chains is the same for all four connection modes available in the ASM330LHB.

Figure 4. **MLC inputs (accelerometer)**

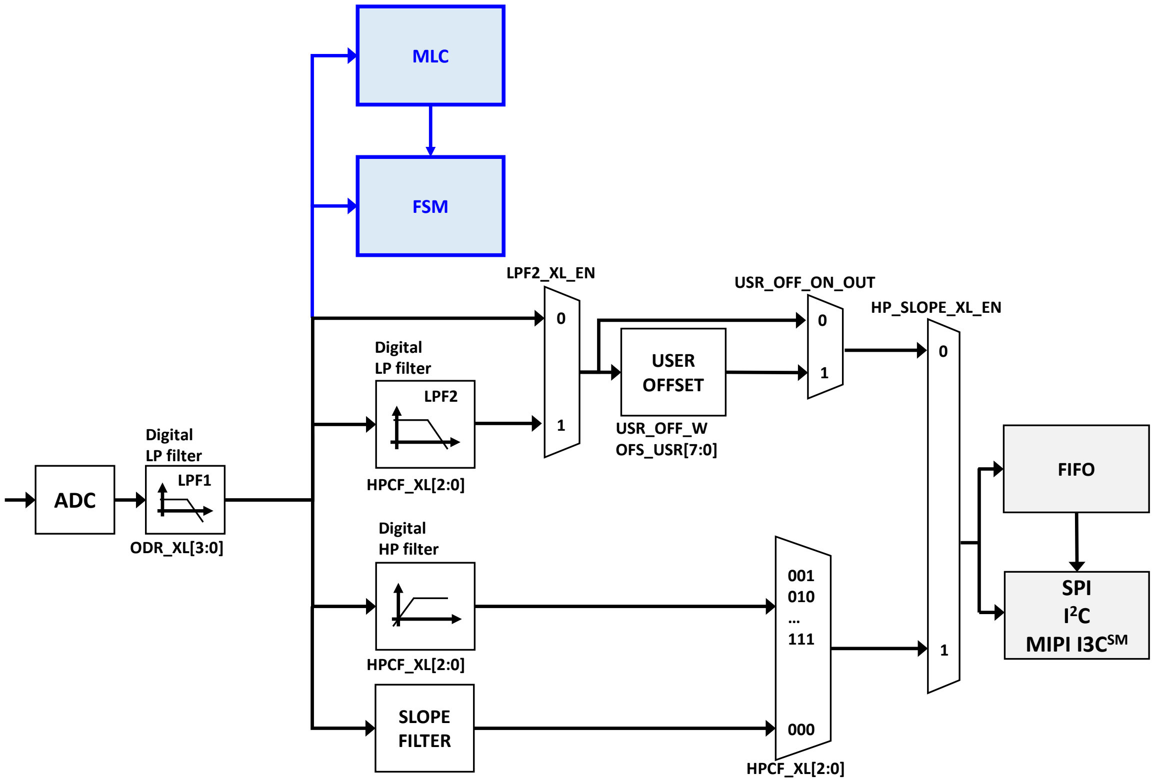
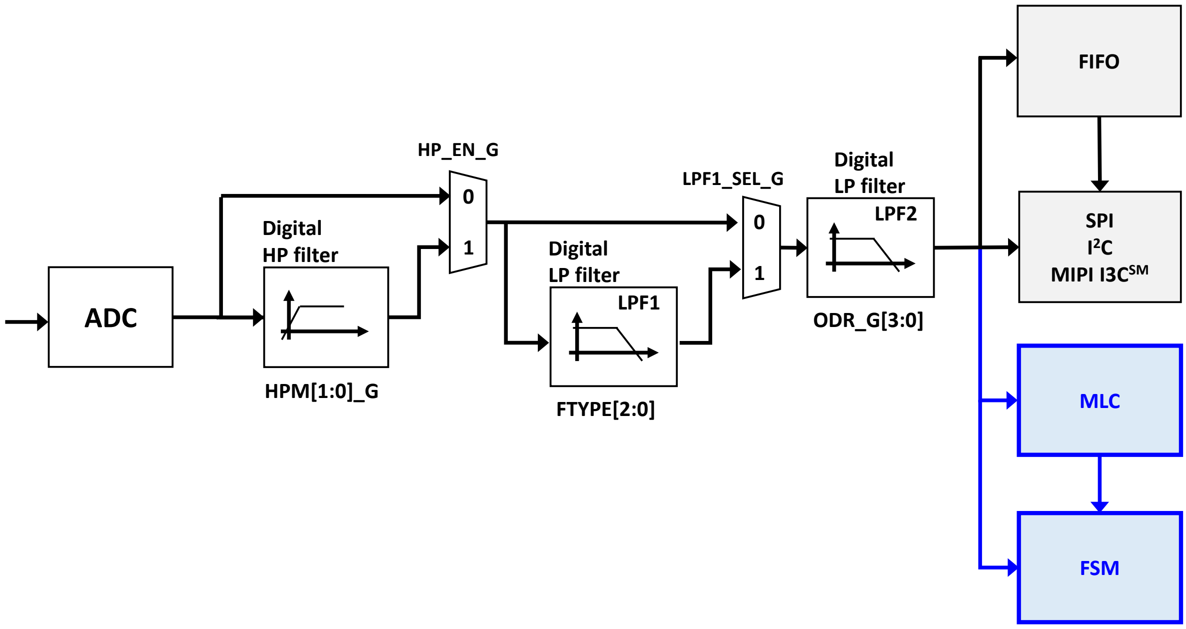


Figure 5. **MLC inputs (gyroscope)**



The rate of the input data must be equal to or higher than the machine learning core data rate configurable through the embedded function register EMB_FUNC_ODR_CFG_C (60h), as described in Table 1.

Example: In an activity recognition algorithm running at 26 Hz, the machine learning core ODR must be selected at 26 Hz, while the sensor ODRs must be equal to or higher than 26 Hz.

The machine learning core uses the following unit conventions:

- Accelerometer data in [*g*]
- Gyroscope data in [rad/sec]

To summarize the machine learning core inputs:

- Accelerometer data conversion factor is automatically handled by the device.
- Gyroscope data conversion factor is automatically handled by the device.

An additional input available for sensor data (accelerometer and gyroscope) is the norm. From the 3-axis data, the machine learning core (in the ASM330LHB) internally computes the norm and the squared norm. These two additional signals can be used as inputs for machine learning processing.

The norm and the squared norm of the input data are computed with the following formulas:

$$V = \sqrt{x^2 + y^2 + z^2}$$

$$V^2 = x^2 + y^2 + z^2$$

Norm and squared norm data can be used in the decision trees in order to guarantee a high level of program customization for the user.

Note: *The data rate for MLC inputs is set through the MLC_ODR bits. If the sensor ODR is higher than MLC_ODR, MLC automatically decimates the input data (without any additional filtering).*

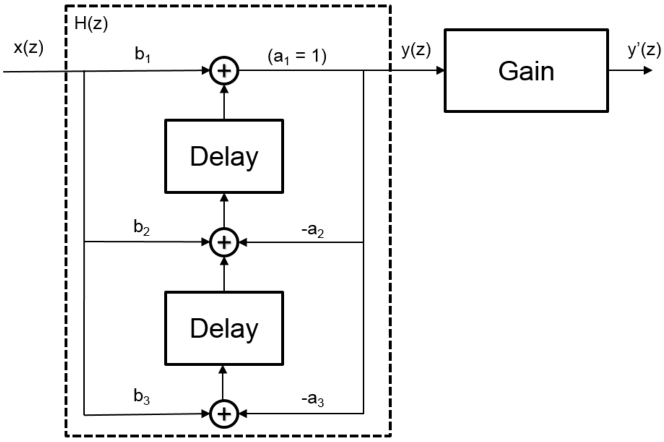
It is recommended to select MLC_ODR equal to the sensor ODR to avoid decimation of the MLC inputs. Selecting MLC_ODR lower than the sensor ODR is also possible, but the different frequency response could lower the accuracy of the MLC solution.

1.2 Filters

The input data seen in the previous section can be filtered by different kinds of filters available in the machine learning core logic. The basic element of the machine learning core filtering is a second order IIR filter, as shown in Figure 6.

Note: The filters available in the MLC block are independent of any other filter available on the device (the filters described in this section are illustrated in the MLC block of Figure 4 and Figure 5).

Figure 6. Filter basic element



The transfer function of the generic IIR 2nd order filter is the following:

$$H\left(z\right) = \frac{b_1 + b_2 z^{-1} + b_3 z^{-2}}{1 + a_2 z^{-1} + a_3 z^{-2}}$$

From Figure 1, the outputs can be defined as:

$$y(z) = H(z) \cdot x(z)$$
$$y'(z) = y(z) \cdot Gain$$

To optimize memory usage, the machine learning core has default coefficients for the different kinds of filters (high-pass, band-pass, IIR1, IIR2). The machine learning core tool helps in configuring the filter by asking for the filter coefficients needed after selecting the kind of filter. The following table shows the default values and the configurable values for the coefficients, depending on the filter type chosen. By setting different coefficients, it is possible to tune the filter for the specific application.

Table 2. Filter coefficients

Filter type / Coefficients	b_1	b_2	b_3	a_2	a_3	Gain
High-pass filter	0.5	-0.5	0	0	0	1
Band-pass filter	1	0	-1	Configurable	Configurable	Configurable
IIR1 filter	Configurable	Configurable	0	Configurable	0	1
IIR2 filter	Configurable	Configurable	Configurable	Configurable	Configurable	1

The filter coefficient values are expressed as half-precision floating-point format: SEEEEEFFFFFFFFFF (S: 1 sign bit; E: 5 exponent bits; F: 10 fraction bits).

1.2.1 Filter coefficients

The IIR filter coefficients can be computed with different tools, including Matlab, Octave and Python. In Matlab, for instance, the following function can be used to generate coefficients for a low-pass filter:

```
[b, a] = butter( N,f_cut / (ODR/2), 'low' )
```

Where:

- N is the order of the IIR filter (1 for IIR1, 2 for IIR2)
- f_cut is the cutoff frequency [Hz] of the filter
- ODR is the machine learning core data rate [Hz]
- 'low' (or 'high') is the kind of filter to be implemented (low-pass or high-pass)

Note: It is possible to configure a high-pass filter with the cutoff at half of the bandwidth (ODR/4) without inserting the coefficients. The machine learning core has some pre-defined coefficients for this configuration.

The following function instead allows generating band-pass filter coefficients through Matlab:

```
[b,a] = butter(1,[f1 f2]/(ODR/2),'bandpass')
```

Note: Since only a2, a3 and gain are configurable for a band-pass filter, the b vector should be normalized by setting gain = b(1). Bandpass filters are generated as first-order filters in Matlab and Python.

Example:

b = [0.2929 0 -0.2929]; a = [1.0 -0.5858 0.4142];

can be written as b = [1 0 -1] and gain = 0.2929.

So the band-pass filter coefficients are:

a2 = -0.5858; a3 = 0.4142; gain = 0.2929.

The following table shows some examples of filter coefficients (most of them considering an ODR of 26 Hz).

When designing high-pass and band-pass IIR filters, consider the stability of the filters in half-precision floating point. The resolution loss can cause some divergence in the signal if the filters are not very stable.

We recommend using first-order IIR filters when the cutoff frequency (normalized) is below 0.02 [2*f_cutoff/ODR] or increasing the cutoff frequency.

Table 3. **Examples of filter coefficients**

Filter type / Coefficients	b_1	b_2	b_3	a_2	a_3	Gain
High-pass IIR1, f_cut = 1 Hz, ODR = 26 Hz	0.891725	-0.891725	-	-0.783450	-	1
High-pass IIR1, f_cut = 2 Hz, ODR = 26 Hz	0.802261	-0.802261	-	-0.604521	-	1
High-pass IIR1, f_cut = 5 Hz, ODR = 26 Hz	0.591628	-0.591628	-	-0.183257	-	1
High-pass IIR1, f_cut = 10 Hz, ODR = 26 Hz	0.274968	-0.274968	-	0.450063	-	1
High-pass IIR2, f_cut = 1 Hz, ODR = 26 Hz	0.8428435	-1.685687	0.8428435	-1.6608344	0.710540	1
High-pass IIR2, f_cut = 2 Hz, ODR = 26 Hz	0.709560	-1.419120	0.709560	-1.332907	0.505334	1
High-pass IIR2, f_cut = 5 Hz, ODR = 26 Hz	0.4077295	-0.815459	0.407730	-0.426937	0.203981	1
High-pass IIR2, f_cut = 10 Hz, ODR = 26 Hz	0.085605	-0.171209	0.085605	1.019146	0.361564	1
Low-pass IIR1, f_cut = 1 Hz, ODR = 26 Hz	0.108275	0.108275	-	-0.783450	-	1
Low-pass IIR1, f_cut = 2 Hz, ODR = 26 Hz	0.197739	0.197739	-	-0.604521	-	1
Low-pass IIR1, f_cut = 5 Hz, ODR = 26 Hz	0.408372	0.408372	-	-0.183257	-	1
Low-pass IIR1, f_cut = 10 Hz, ODR = 26 Hz	0.725032	0.725032	-	0.450063	-	1
Low-pass IIR2, f_cut = 1 Hz, ODR = 26 Hz	0.012426	0.024853	0.012426	-1.660834	0.710540	1
Low-pass IIR2, f_cut = 2 Hz, ODR = 26 Hz	0.043107	0.086213	0.043107	-1.332907	0.505333	1
Low-pass IIR2, f_cut = 5 Hz, ODR = 26 Hz	0.194261	0.388522	0.194261	-0.426937	0.203981	1
Low-pass IIR2, f_cut = 10 Hz, ODR = 26 Hz	0.595178	1.190355	0.595178	1.019146	0.361564	1
Band-pass IIR2, f1 = 1.5 Hz, f2 = 5 Hz, ODR = 26 Hz	0.310375	0	-0.310375	-1.069500	0.379250	1
Band-pass IIR2, f1 = 0.2 Hz, f2 = 1 Hz, ODR = 100 Hz	0.0236	0	-0.0236	-1.9521	0.9528	1

1.3 Features

The features are the statistical parameters computed from the machine learning core inputs. The machine learning core inputs which can be used for features computation are:

- The sensor input data which includes
  - Sensor data from the X, Y, Z axes (for example, Acc_X, Acc_Y, Acc_Z, Gyro_X, Gyro_Y, Gyro_Z)
  - Norm and squared norm signals of sensor data (Acc_V, Acc_V2, Gyro_V, Gyro_V2)
- The filtered data (for example, high-pass on Acc_Z, band-pass on Acc_V2, and so forth)

All the features are computed within a defined time window, which is also called “window length” since it is expressed as the number of samples. The size of the window has to be determined by the user and is very important for the machine learning processing, since all the statistical parameters in the decision tree are evaluated in this time window. It is not a moving window, features are computed just once for every WL sample (where WL is the size of the window).

The window length can have values from 1 to 255 samples. The choice of the window length value depends on the MLC data rate (MLC_ODR bits in the embedded function register EMB_FUNC_ODR_CFG_C (60h)), which introduces a latency for the generation of the machine learning core result, and in the specific application or algorithm. In an activity recognition algorithm for instance, it can be decided to compute the features every 2 or 3 seconds, which means that considering sensors running at 26 Hz, the window length should be around 50 or 75 samples respectively.

Some of the feaures in the machine learning core require some additional parameters for the evaluation (for example, an additional threshold). The following table shows all the features available in the machine learning core including additional parameters.

Note: The maximum number of features which can be configured in the MLC is 63. Feature values are limited to the range ±65536.

Table 4. Features

Feature	Additional parameter
MEAN	-
VARIANCE	-
ENERGY	-
PEAK-TO-PEAK	-
ZERO-CROSSING	Threshold
POSITIVE ZERO-CROSSING	Threshold
NEGATIVE ZERO-CROSSING	Threshold
PEAK DETECTOR	Threshold
POSITIVE PEAK DETECTOR	Threshold
NEGATIVE PEAK DETECTOR	Threshold
MINIMUM	-
MAXIMUM	-

1.3.1 Mean

The feature “*Mean*” computes the average of the selected input (*I*) in the defined time window (*WL*) with the following formula:

$$Mean = \frac{1}{WL} \sum_{k=0}^{WL-1} I_k$$

1.3.2 Variance

The feature “*Variance*” computes the variance of the selected input (*I*) in the defined time window (*WL*) with the following formula:

$$Variance = \left(\frac{\sum_{k=0}^{WL-1} {I_k}^2}{WL}\right) - \left(\frac{\sum_{k=0}^{WL-1} I_k}{WL}\right)^2$$

1.3.3 Energy

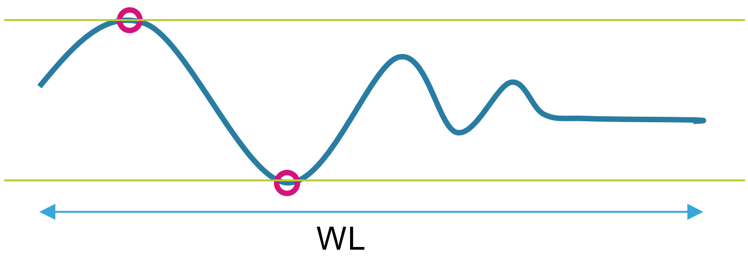
The feature “*Energy*” computes the energy of the selected input (*I*) in the defined time window (*WL*) with the following formula:

$$Energy = \sum_{k=0}^{WL-1} {I_k}^2$$

1.3.4 Peak-to-peak

The feature “*Peak-to-peak*” computes the maximum peak-to-peak value of the selected input in the defined time window.

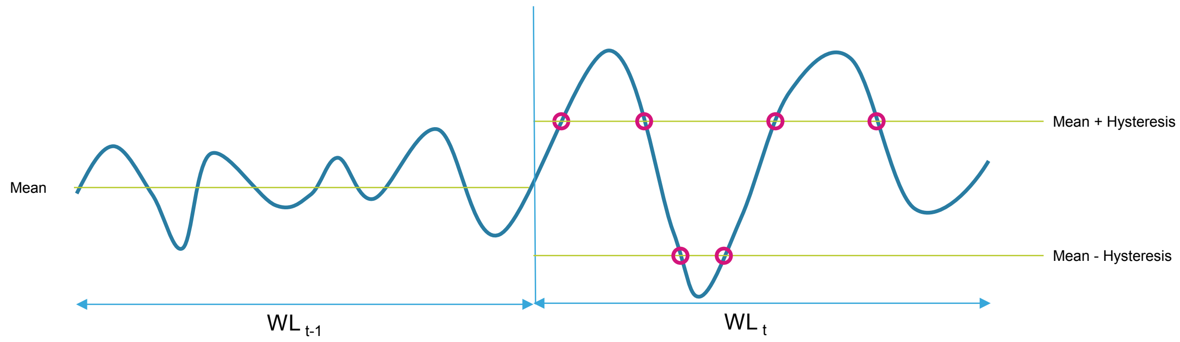
Figure 7. Peak-to-peak



1.3.5 Zero-crossing

The feature "*Zero-crossing*" computes the number of times the selected input crosses a certain threshold. This internal threshold is defined as the sum between the average value computed in the previous window (feature "Mean") and hysteresis defined by the user.

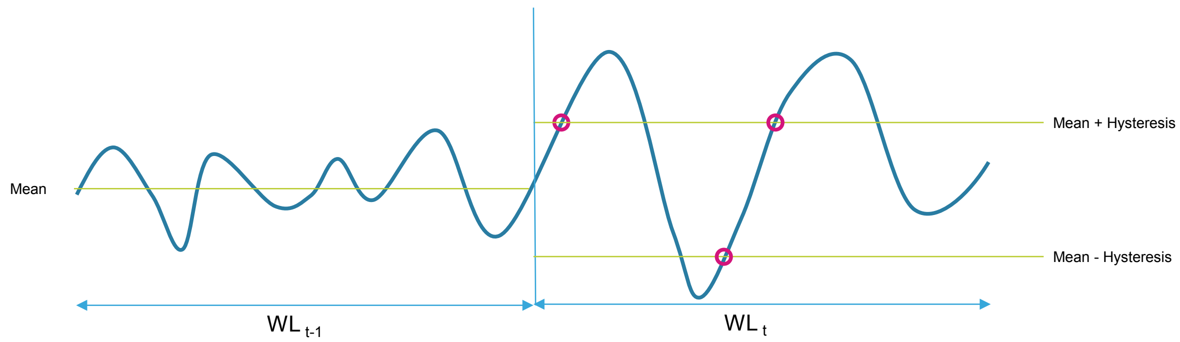
Figure 8. Zero-crossing



1.3.6 Positive zero-crossing

The feature "*Positive zero-crossing*" computes the number of times the selected input crosses a certain threshold. This internal threshold is defined as the sum between the average value computed in the previous window (feature "Mean") and hysteresis defined by the user. Only the transitions with positive slopes are considered for this feature.

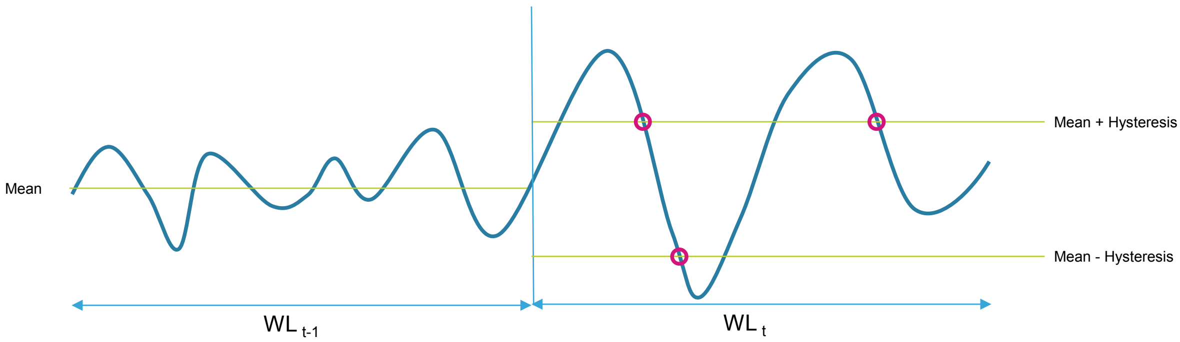
Figure 9. Positive zero-crossing



1.3.7 Negative zero-crossing

The feature "*Negative zero-crossing*"computes the number of times the selected input crosses a certain threshold. This internal threshold is defined as the sum between the average value computed in the previous window (feature "Mean") and hysteresis defined by the user. Only the transitions with negative slopes are considered for this feature.

Figure 10. Negative zero-crossing



1.3.8 Peak detector

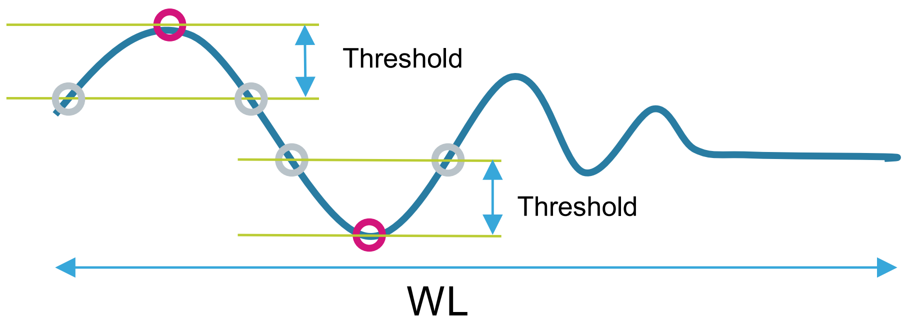
The feature "*Peak detector*" counts the number of peaks (positive and negative) of the selected input in the defined time window.

A threshold has to be defined by the user for this feature, and a buffer of three values is considered for the evaluation. If the second value of the three values buffer is higher (or lower) than the other two values of a selected threshold, the number of peaks is increased.

The buffer of three values considered for the computation of this feature is a moving buffer inside the time window.

The following figure shows an example of the computation of this feature, where two peaks (one positive and negative) have been detected in the time window.

Figure 11. Peak detector



1.3.9 Positive peak detector

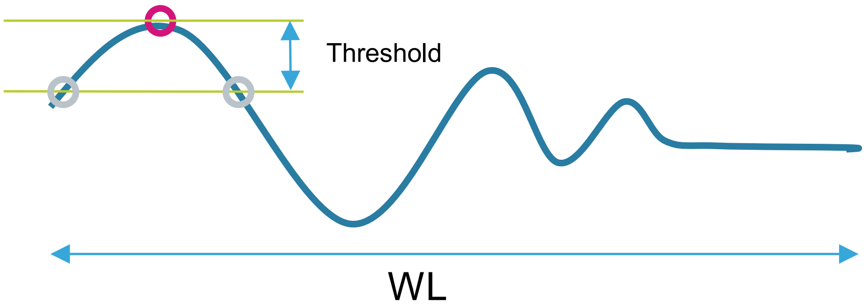
The feature "*Positive peak detector*" counts the number of positive peaks of the selected input in the defined time window.

A threshold has to be defined by the user for this feature, and a buffer of three values is considered for the evaluation. If the second value of the three values buffer is higher than the other two values of a selected threshold, the number of peaks is increased.

The buffer of three values considered for the computation of this feature is a moving buffer inside the time window.

The following figure shows an example of the computation of this feature, where just one peak (positive) has been detected in the time window.

Figure 12. **Positive peak detector**



1.3.10 Negative peak detector

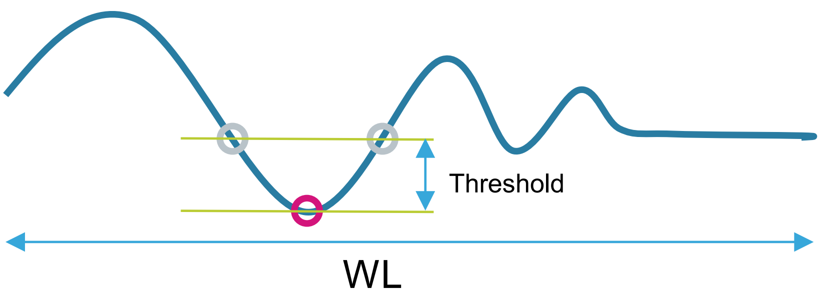
The feature "*Negative peak detector*" counts the number of negative peaks of the selected input in the defined time window.

A threshold has to be defined by the user for this feature, and a buffer of three values is considered for the evaluation. If the second value of the three values buffer is lower than the other two values of a selected threshold, the number of peaks is increased.

The buffer of three values considered for the computation of this feature is a moving buffer inside the time window.

The following figure shows an example of the computation of this feature, where just one peak (negative) has been detected in the time window.

Figure 13. **Negative peak detector**

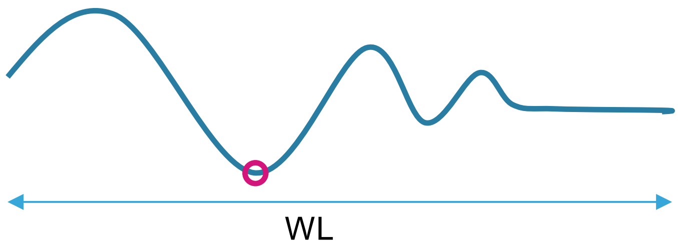


1.3.11 Minimum

The feature "*Minimum*" computes the minimum value of the selected input in the defined time window.

The following figure shows an example of minimum in the time window.

Figure 14. **Minimum**

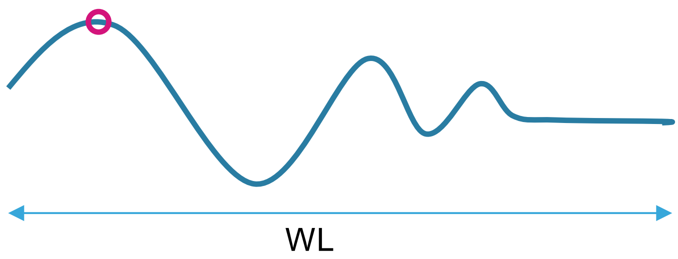


1.3.12 Maximum

The feature "*Maximum*" computes the maximum value of the selected input in the defined time window.

The following figure shows an example of maximum in the time window.

Figure 15. **Maximum**



1.3.13 Selection of features

The selection of the features to be used for the machine learning core configuration depends on the specific application.

Considering that the use of too many features may lead to overfitting and too large decision trees, it is recommended to start first by selecting the four most common features:

- Mean
- Variance
- Energy
- Peak-to-peak

If the performance is not good with these features, and in order to improve the accuracy, other features can be considered to better separate the classes.

Input data for the features calculation (from the accelerometer, gyroscope) and axes (for example, X, Y, Z, V) have to be chosen according to the specific application as well. Some classes are strongly correlated with sensor orientation (that is, applications which use the device carry position), so it is better to use individual axis (X, Y, Z). Other classes (like walking) are independent of orientation, so it is better to use the norm (V or V2).

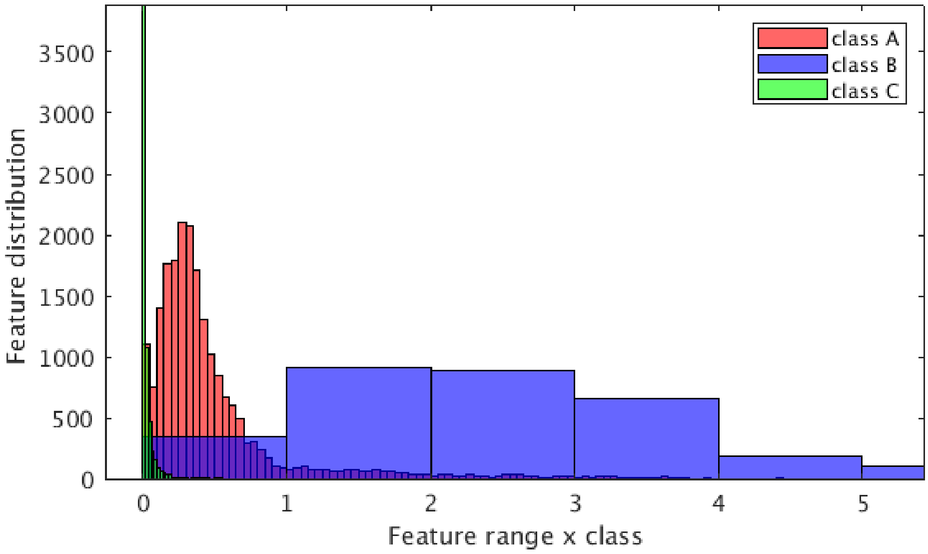
Sometimes the basic features (mean, variance, energy, and so forth) might not help in distinguishing the dominating frequency, so embedded digital filters can be enabled to select a specific region of frequency. Using the filtered signal, certain classes may be distinguished more precisely. For instance, if the user is walking, the typical signal is around 1-2 Hz, while if the user is jogging, the typical signal is around 2.5-4 Hz.

The information contribution from a single feature can be evaluated by a measure of how much different classes are separated (from one another). This analysis can be done in a graphical way, by plotting 1D/2D graphs as described in the following examples.

1.3.13.1 *Histogram of a single feature (1D plot)*

The following figure shows a histogram of the computed values of a single feature for three different classes. These three classes are reasonably separated, so an important level of information is expected with this feature. For reference, the computed classification accuracy with this single feature is around 75%.

Figure 16. Distribution of single feature for three different classes



1.3.13.2 *Visualization of two features (2D plot)*

The following figure shows a 2D plot related to a 2-class classification problem with the selection of two features:

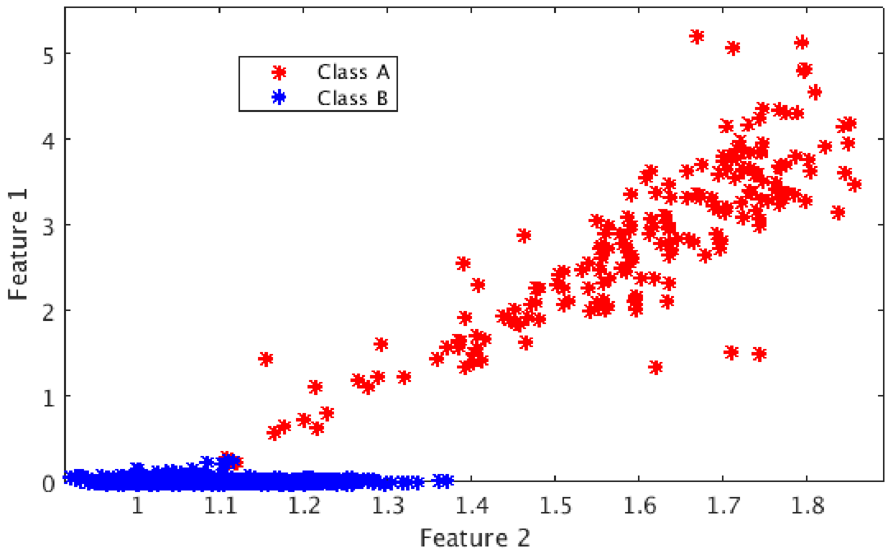
- Feature 1 on the graph vertical axis
- Feature 2 on the graph horizontal axis

In this case, the strict separation between the two classes is evident:

- Class A in red
- Class B in blue

A good information contribution can be obtained by combining the two features. For reference, the classification accuracy obtained with this example is more than 95%.

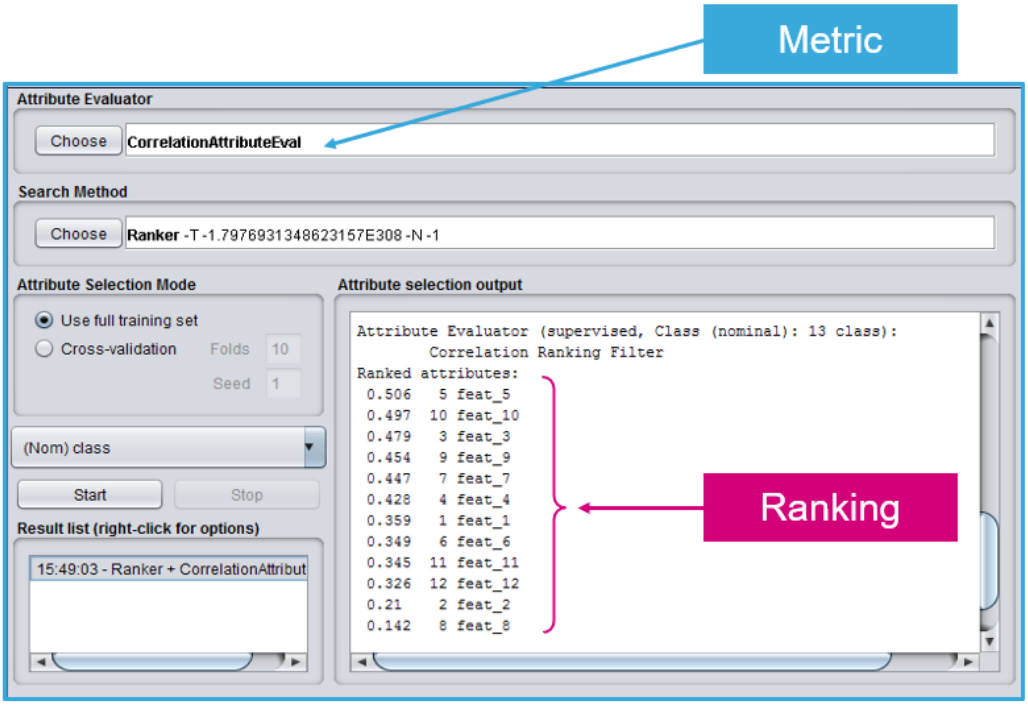
Figure 17. Visualization of two features and two classes



1.3.13.3 *Ranking of features*

Different machine learning tools offer automated methods to order features in terms of the information contribution. This form of output ranking is based on criteria/metrics such as correlation, information gain, probabilistic distance, entropy and more. An example is given by Weka, which automatically handles the calculations needed to generate optimal decision trees as indicated in the figure below.

Figure 18. Ranking from automated output tool



Note that different features could share the same information contribution. This can be evaluated again by visualizing the single feature or by checking the accuracy obtained with the subset of features taken one-by-one, and together, as explained in previous sections.

A final consideration can be done on the number of features which have been selected. In general, the higher the number of features selected:

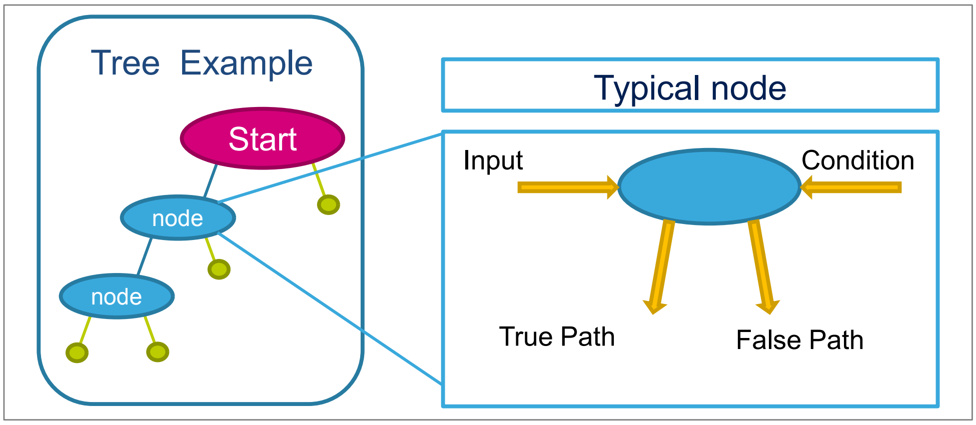
- The higher the risk of overfitting
- The larger the size of the resulting decision tree

1.4 Decision tree

The decision tree is the predictive model built from the training data which can be stored in the ASM330LHB. The training data are the data logs acquired for each class to be recognized (in the activity recognition example the classes might be walking, jogging, driving, and so forth).

The outputs of the computation blocks described in the previous sections are the inputs of the decision tree. Each node of the decision tree contains a condition, where a feature is evaluated with a certain threshold. If the condition is true, the next node in the true path is evaluated. If the condition is false, the next node in the false path is evaluated. The status of the decision tree evolves node by node until a result is found. The result of the decision tree is one of the classes defined at the beginning of the data collection.

Figure 19. Decision tree node



The decision tree generates a new result every time window (the parameter "window length" set by the user for the computation of the features). Window length is expressed as a number of samples. The time window can be obtained by dividing the number of samples by the data rate chosen for MLC (MLC_ODR):

Time window = Window length / MLC_ODR

For instance, selecting 104 samples for the window length and 104 Hz for the MLC data rate, the obtained time window is:

Time window = 104 samples / 104 Hz = 1 second

The decision tree results can also be filtered by an additional (optional) filter called "meta-classifier", which is described in Section 1.5 Meta-classifier.

The machine learning core results (decision tree results filtered or not filtered) are accessible through dedicated registers in the embedded advanced features page 1 of the ASM330LHB registers (as shown in Table 5). These registers can be countinuously read (polled) to check the decision tree outputs. The register MLC_STATUS_MAINPAGE (38h) contains the interrupt status bits of the 8 possible decision trees. These bits are automatically set to 1 when the corresponding decision tree value changes. Furthermore, the interrupt status signal generated using these bits can also be driven to the INT1 pin by setting the MLC_INT1 (0Dh) register, or to the INT2 pin by setting the MLC_INT2 (11h) register (Table 6). Using the interrupt signals, an MCU performing other tasks or sleeping (to save power), can be awakened when the machine learning core result has changed.

The machine learning core interrupt signal is pulsed by default. The duration of the pulsed interrupt is defined by the fastest ODR among the machine learning core, finite state machine and sensor ODRs:

```
interrupt pulse duration = 1 / max(MLC_ODR, FSM_ODR, XL_ODR, GYRO_ODR)
```

The machine learning core interrupt signal can also be set latched through the bit EMB_FUNC_LIR in the embedded function register PAGE_RW (17h).

Table 5. Decision tree results

Register	Content
MLC0_SRC (70h)	Result of decision tree 1
MLC1_SRC (71h)	Result of decision tree 2
MLC2_SRC (72h)	Result of decision tree 3
MLC3_SRC (73h)	Result of decision tree 4
MLC4_SRC (74h)	Result of decision tree 5
MLC5_SRC (75h)	Result of decision tree 6
MLC6_SRC (76h)	Result of decision tree 7
MLC7_SRC (77h)	Result of decision tree 8

Table 6. Decision tree interrupts

Register	Content
MLC_STATUS_MAINPAGE (38h)	Contains interrupt status bits for changes in the decision tree result
MLC_STATUS (15h)	Contains interrupt status bits for changes in the decision tree result
MLC_INT1 (0Dh)	Allows routing the interrupt status bits for decision trees to INT1 pin(1)
MLC_INT2 (11h)	Allows routing the interrupt status bits for decision trees to INT2 pin(2)

1. *Routing is established if the INT1_EMB_FUNC bit of MD1_CFG (5Eh) is set to 1.*
2. *Routing is established if the INT2_EMB_FUNC bit of MD2_CFG (5Fh) is set to 1.*

1.4.1 Decision tree limitations in the ASM330LHB

The ASM330LHB has limited resources for the machine learning core in terms of number of decision trees, size of the trees, and number of decision tree results.

Up to 8 different decision trees can be stored in the ASM330LHB, but the sum of the number of nodes for all the decision trees must not exceed 512 (*). Every decision tree can have up to 256 results in the ASM330LHB.

(*) This number might also be limited by the number of features and filters configured. In general, if using few filters and features, there is no further limitation on the size of the decision tree. However, when using many filters and features, the maximum number of nodes for the decision trees is slightly limited. For instance, if the number of filters configured is 10 and the number of features configured is 50, the maximum number of nodes might be reduced by 100. The tool informs the user of the available nodes for the decision tree.

The table below summarizes the limitations of the ASM330LHB.

Table 7. Decision tree limitations in the ASM330LHB

	ASM330LHB
Maximum number of decision trees	8
Maximum number of nodes (total number for all the decision trees)	512 (*)
Maximum number of results per decision tree	256

Note: When using multiple decision trees, all the parameters described in the previous sections (inputs, filters, features computed in the time window, the time window itself, and also the data rates) are common for all the decision trees.

1.5 Meta-classifier

A meta-classifier is a filter on the outputs of the decision tree. The meta-classifier uses some internal counters in order to filter the decision tree outputs.

Decision tree outputs can be divided in subgroups (for example, similar classes can be managed in the same subgroup). An internal counter is available for all the subgroups of the decision tree outputs. The counter for the specific subgroup is increased when the result of the decision tree is one of the classes in the subgroup and it is decreased otherwise. When the counter reaches a defined value, which is called “end counter” (set by the user), the output of the machine learning core is updated. Values allowed for end counters are from 0 to 14.

Table 8. Meta-classifier example

Decision tree result	A	A	A	B	A	B	B	B	A	B	B	B	A	A	A
Counter A (End counter = 3)	1	2	**3**	2	3	2	1	0	1	0	0	0	1	2	**3**
Counter B (End counter = 4)	0	0	0	1	0	1	2	3	2	3	**4**	5	4	3	2
Machine learning core result (including meta-classifier)	x	x	**A**	**A**	**A**	**A**	**A**	**A**	**A**	**A**	**B**	**B**	**B**	**B**	**A**

The previous table shows the effect of filtering the decision tree outputs through a meta-classifier. The first line of the table contains the outputs of the decision tree before the meta-classifier. Counter A and counter B are the internal counters for the two decision tree results (“A” and “B”). In the activity recognition example, the result “A” might be walking and the result “B” jogging. When the internal counter “A” reaches the value 3 (which is the end counter for counter “A”), there is a transition to result “A”. When the internal counter “B” reaches value 4, there is a transition to result “B”.

The purpose of the meta-classifier is to reduce the false positives, in order to avoid generating an output which is still not stable, and to reduce the transitions on the decision tree result.

1.5.1 Meta-classifier limitations in the ASM330LHB

The meta-classifier has a limited number of subgroups, 8 subgroups can be used in the ASM330LHB. Similar classes may need to be grouped in the same subgroup to use the meta-classifier.

Table 9. Meta-classifier limitations in the ASM330LHB

	ASM330LHB
Maximum number of results per decision tree	256
Result subgroups for meta-classifier per decision tree	8

Note: Multiple meta-classifiers can be configured. One meta-classifier is available for any decision tree configured in the machine learning core.

1.6 Finite state machine interface

The ASM330LHB also provides a configurable finite state machine which is suitable for deductive algorithms and in particular gesture recognition.

Finite state machines and decision trees can be combined to work together in order to enhance the accuracy of motion detection.

The decision tree results generated by the machine learning core can be checked by the finite state machine available in the ASM330LHB; this is possible through the condition CHKDT (described in the finite state machine application note).

2 Machine learning core tools

The machine learning core programmability in the device is allowed through a dedicated tool, available as an extension of the Unico GUI.

2.1 Unico GUI

Unico is the graphical user interface for all the MEMS sensor demonstration boards available in the STMicroelectronics portfolio. It has the possibility to interact with a motherboard based on the STM32 microcontroller (professional MEMS tool), which enables the communication between the MEMS sensor and the PC GUI. Unico also has the possibility to run offline, without a motherboard connected to the PC.

Details of the professional MEMS tool board can be found on www.st.com at STEVAL-MKI109V3.

Unico GUI is available in three software packages for the three operating systems supported.

- Windows
  - STSW-MKI109W
- Linux
  - STSW-MKI109L
- Mac OS X
  - STSW-MKI109M

Unico GUI allows visualization of sensor outputs in both graphical and numerical format and allows the user to save or generally manage data coming from the device.

Unico allows access to the MEMS sensor registers, enabling fast prototyping of register setup and easy testing of the configuration directly on the device. It is possible to save the current register configuration in a text file (with .ucf extension) and load a configuration from an existing file. In this way, the sensor can be reprogrammed in few seconds.

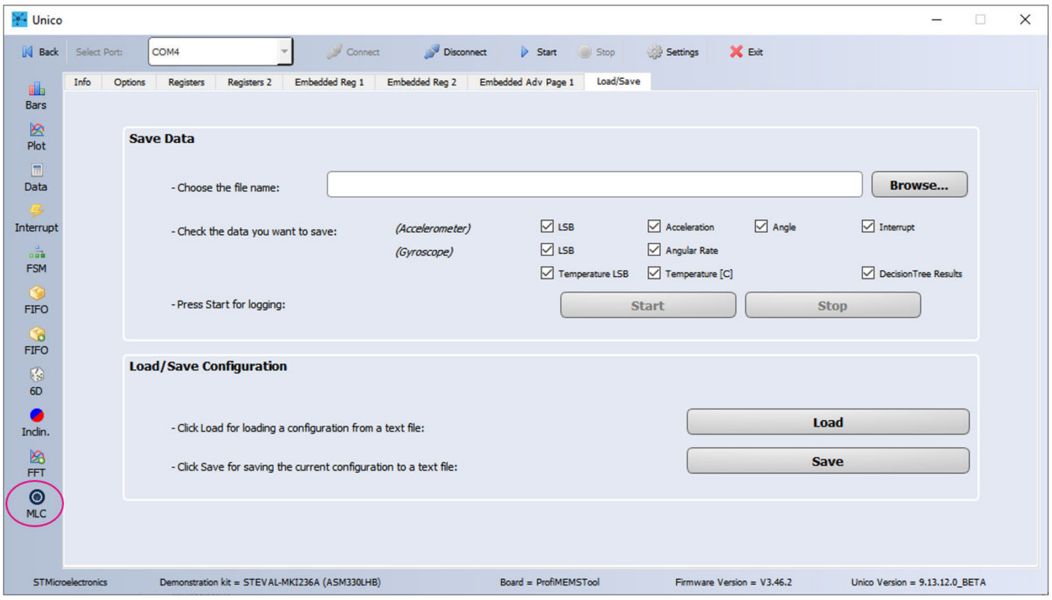
The machine learning core tool available in the Unico GUI abstracts the process of register configuration by automatically generating configuration files for the device. The user just needs to set some parameters in the GUI and by clicking a few buttons, the configuration file is already available. From these configuration files, the user can create his own library of configurations for the device.

Since the machine learning approach requires the collection of data logs, they can be acquired through the [**Load/Save**] tab of Unico (Figure 20). For the accelerometer, the checkbox [**Acceleration**] allows saving data in [m*g*]. For the gyroscope, the checkbox [**Angular rate**] allows saving data in [dps].

*Note: When logging data, the [**Start**] and [**Stop**] buttons (in the [**Load/Save**] tab of Unico) must be used properly in order to avoid logging incorrect data at the beginning or at the end of the acquisition. For instance, when logging a data pattern for the class "walking", the user should start walking before pressing the button [**Start**] and stop walking after pressing the button [**Stop**]. It is important to select the correct ODR for data logging. If the final MLC ODR (for example, 26 Hz) is already defined, it is recommended to use the same ODR for data logging (ODR 26 Hz). If the MLC ODR is not defined, it is recommended to log the data at ODR 104 Hz (which is the maximum ODR for MLC), and then downsample the data if needed. Depending on the algorithm to be implemented, different data logs are needed (at least one per class to use the supervised machine learning approach). It is recommended to have different data logs for each class (for example, 30 data logs per class) in order to capture some diversity or variation, which can be expected in the final application (for example, different users, different tests, or different conditions).*

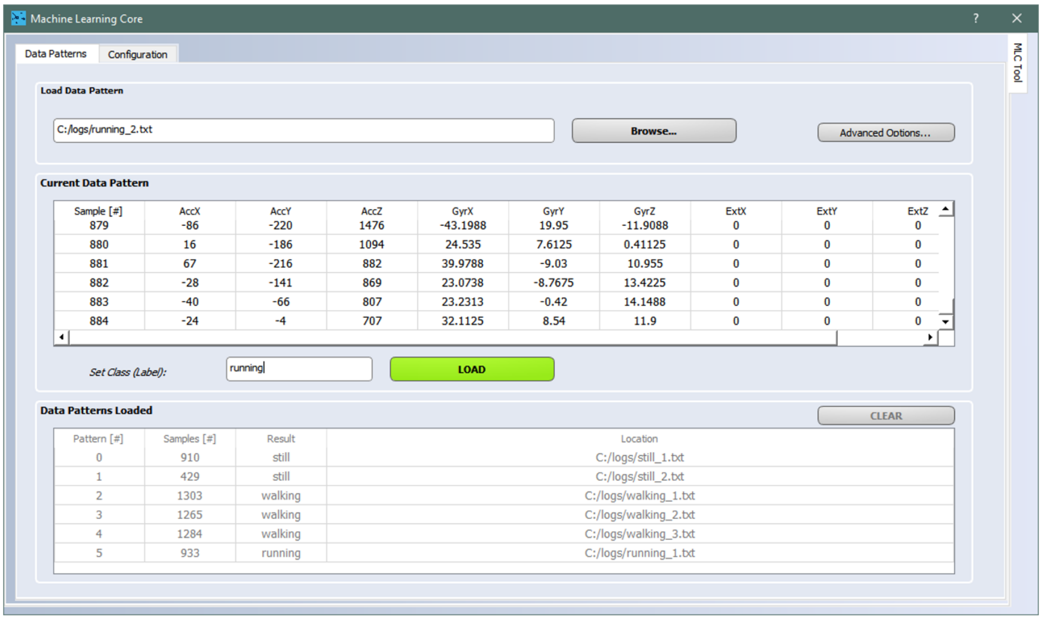
If using Unico GUI offline (without connecting the motherboard to the PC), the user, who has already acquired the data logs, can directly upload them to generate a machine learning core configuration.

Figure 20. **Unico GUI**



The collected data logs can then be loaded in the machine learning core tool of Unico, available on the left side of the GUI, by using the [**Data Patterns**] tab (Figure 21). An expected result must be assigned to each data pattern loaded (for instance, in the activity recognition algorithm, the results might be: still, walking, jogging, and so forth). This assignment is also called "data labeling". The label has to be a set of characters including only letters and numbers (no special characters and no spaces). It is also possible to load a group of data patterns (by multiple selections in the folder where the files are located) and assign the label just once for all the files selected.

Figure 21. **Machine learning core tool - data patterns**



The unit of measurement for the data expected in the [**Data Patterns**] tab of the machine learning core tool are:

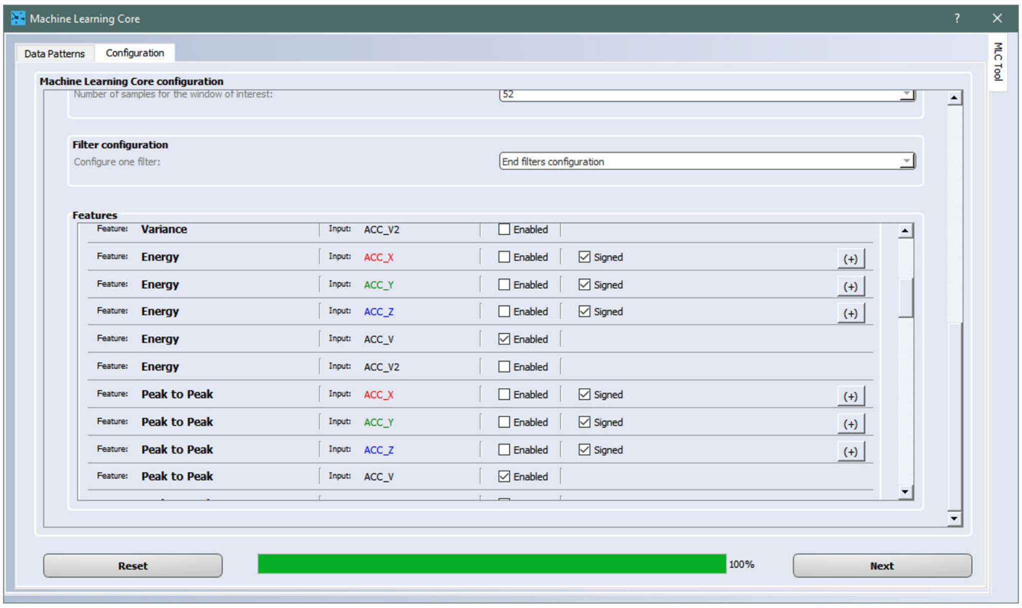
- [mg] (or alternatively [g]) for the accelerometer
- [dps] (or alternatively [mdps]) for the gyroscope

The conversion from [m*g*] to [*g*] for the accelerometer, and [dps] to [rad/s] for the gyroscope, is automatically managed internally by the machine learning core tool, to allow the machine learning core logic to work with the correct data ([*g*] and [rad/s]).

In the configuration tab of the machine learning core tool (Figure 22), all the parameters of the machine learning core (such as ODR, full scales, window length, filters, features, meta-classifier) can be configured. The tool allows selecting multiple filters that can be applied to the raw data, and multiple features to be computed from the input data or from the filtered data. The features computed are the attributes of the decision tree.

When the board is connected and the device already configured, the tool automatically suggests ODRs and full scales (for accelerometer and gyroscope) according to the current device configuration.

Figure 22. Machine learning core tool - configuration



The [**Configuration**] tab of the machine learning core tool generates an attribute-relation file (ARFF), which is the starting point for the decision tree generation process. The decision tree can be generated by different machine learning tools (Section 2.2).

Once the decision tree has been generated, it can be uploaded to the machine learning core tool in Unico to complete the generation of the register configuration for the ASM330LHB.

The Unico GUI, by accessing the sensor registers, can read the status of the decision tree outputs, visualize them together with sensor data, and make it possible to log all the data (sensor outputs and decision tree outputs) together in the same text file.

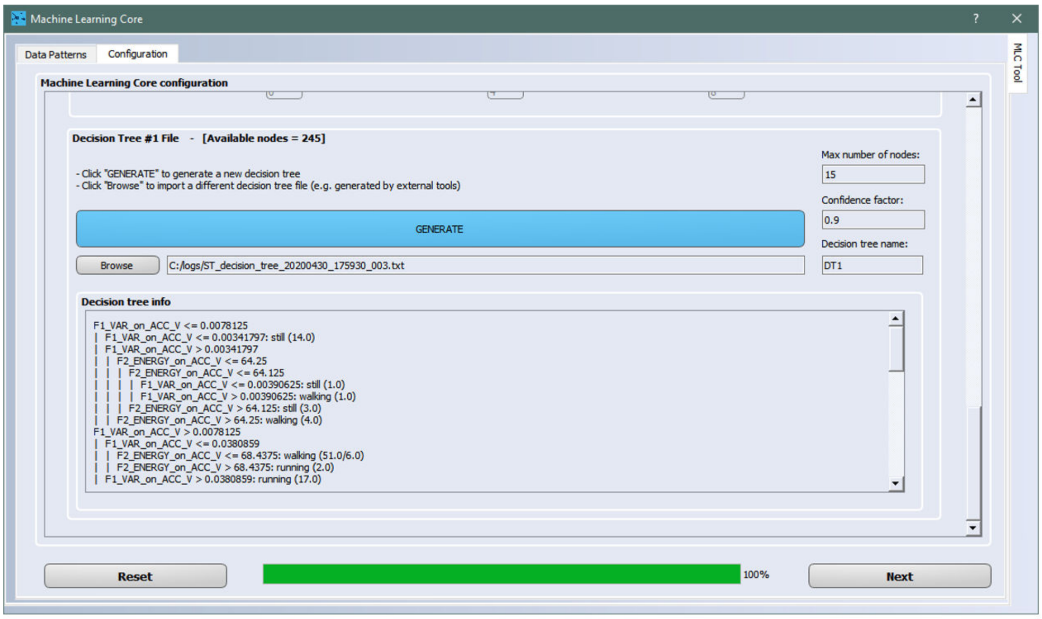
2.2 Decision tree generation

Unico (starting from version 9.8) is able to automatically generate the decision tree, as shown in the following figure.

Two parameters can be fine-tuned when starting the decision tree generation in Unico:

- The maximum number of nodes, which can be set to make sure the generated tree can fit in the MLC configuration;
- The confidence factor, for controlling decision tree pruning (by lowering this number, overfitting can be reduced).

Figure 23. **Decision tree generation in Unico**



Besides Unico, there are other external machine learning tools able to generate decision trees and some of them are supported by Unico.

One of the most frequently used tools is Weka, software developed by the University of Waikato (more details about this software can be found in Appendix A). Other alternative tools are: RapidMiner (Appendix B), Matlab (Appendix C), Python (Appendix D).

Weka is able to generate a decision tree starting from an attribute-relation file (ARFF). Through Weka it is possible to evaluate which attributes are good for the decision tree, and different decision tree configurations can be implemented by changing all the parameters available in Weka. Figure 24 and Figure 25 show the [**Preprocess**] and [**Classify**] tabs of Weka which allow evaluating the attributes and generating the decision tree.

Figure 24. **Weka preprocess**

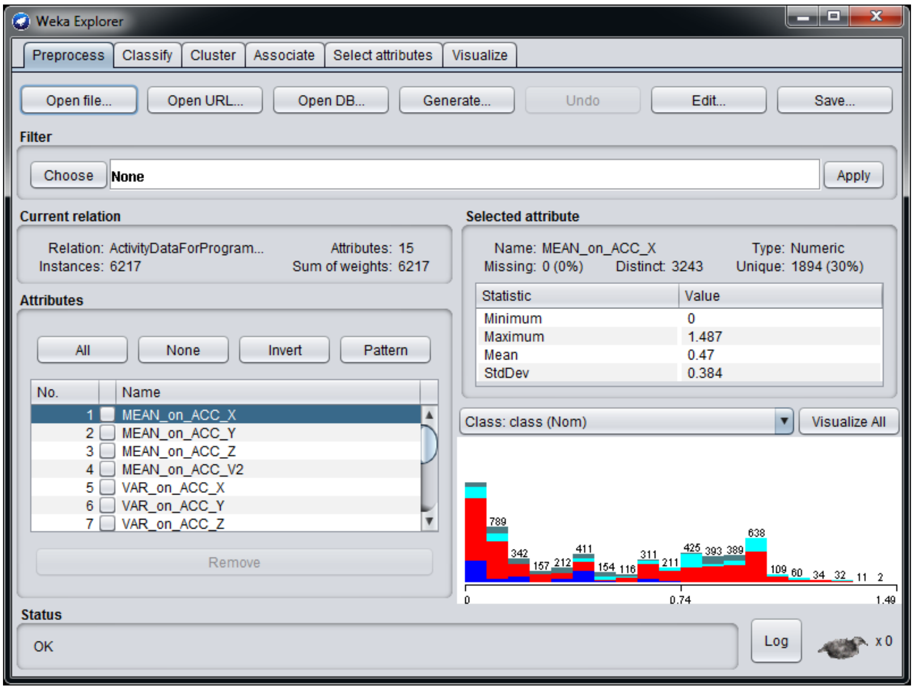
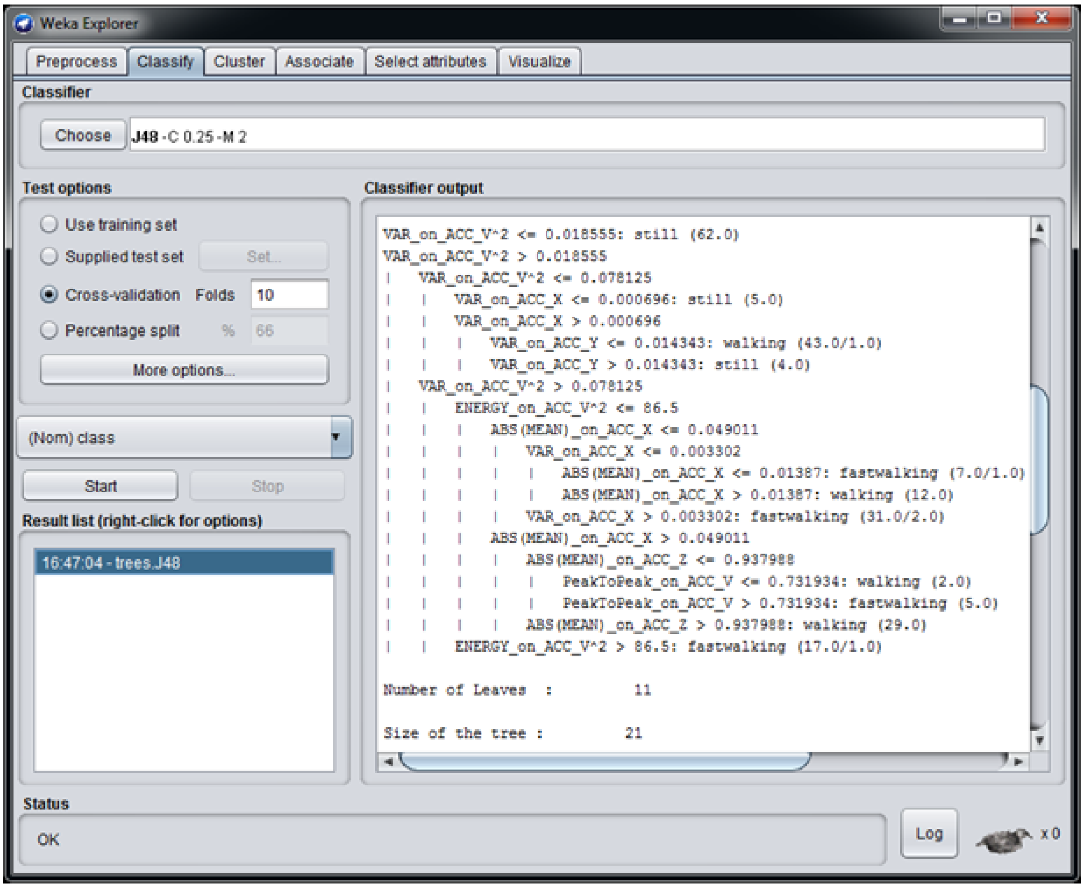


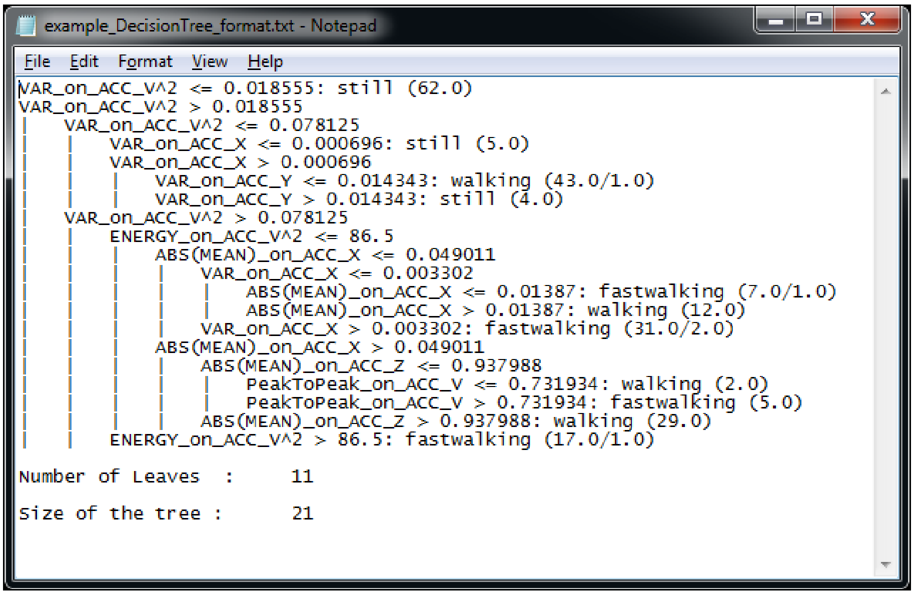
Figure 25. **Weka classify**



Once the decision tree has been generated, it can be uploaded to the machine learning core tool in Unico, to complete the generation of the register configuration for the ASM330LHB.

The machine learning core tool in Unico accepts as input the decision tree files in a textual format (.txt). The textual file must contain the decision tree in the Weka J48 format (an example of a decision tree is shown in Figure 26). From the Weka [**Classifier output**] (Figure 25), the decision tree has to be selected starting from the first line (first node) or in the RapidMiner format (Appendix B). The last two rows (number of leaves and size of the tree) are optional. The selected output from Weka has to be copied to a text file.

Figure 26. **Decision tree format**




```
VAR_on_ACC_V^2 <= 0.018555: still (62.0)
VAR_on_ACC_V^2 > 0.018555
|   VAR_on_ACC_V^2 <= 0.078125
|   |   VAR_on_ACC_X <= 0.000696: still (5.0)
|   |   VAR_on_ACC_X > 0.000696
|   |   |   VAR_on_ACC_Y <= 0.014343: walking (43.0/1.0)
|   |   |   VAR_on_ACC_Y > 0.014343: still (4.0)
|   VAR_on_ACC_V^2 > 0.078125
|   |   ENERGY_on_ACC_V^2 <= 86.5
|   |   |   ABS(MEAN)_on_ACC_X <= 0.049011
|   |   |   |   VAR_on_ACC_X <= 0.003302
|   |   |   |   |   ABS(MEAN)_on_ACC_X <= 0.01387: fastwalking (7.0/1.0)
|   |   |   |   |   ABS(MEAN)_on_ACC_X > 0.01387: walking (12.0)
|   |   |   |   VAR_on_ACC_X > 0.003302: fastwalking (31.0/2.0)
|   |   |   ABS(MEAN)_on_ACC_X > 0.049011
|   |   |   |   ABS(MEAN)_on_ACC_Z <= 0.937988
|   |   |   |   |   PeakToPeak_on_ACC_V <= 0.731934: walking (2.0)
|   |   |   |   |   PeakToPeak_on_ACC_V > 0.731934: fastwalking (5.0)
|   |   |   |   ABS(MEAN)_on_ACC_Z > 0.937988: walking (29.0)
|   |   ENERGY_on_ACC_V^2 > 86.5: fastwalking (17.0/1.0)

Number of Leaves  :     11

Size of the tree :      21
```

If the decision tree has been generated from a different tool, the format must be converted to the Weka J48 format (or to the RapidMiner format) in order to allow the machine learning core tool in Unico to read the decision tree correctly.

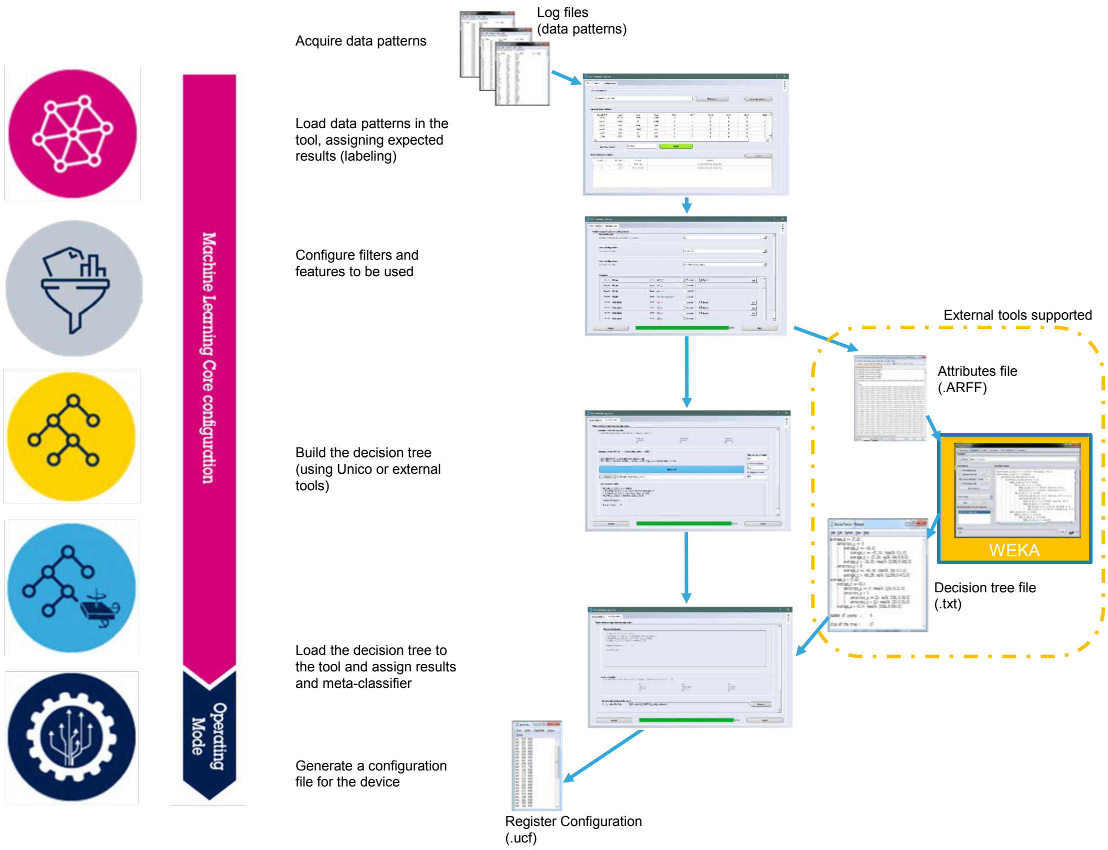
2.3 Configuration procedure

Figure 27 shows the whole procedure of the machine learning processing, from the data patterns to the generation of a register setting for the device (ASM330LHB).

As seen in Section 2.1 Unico GUI, the data patterns can be acquired in the [**Load/Save**] tab of the Unico GUI. If this is not possible or if the user wants to use some different data patterns, they can still be uploaded in the machine learning core tool of Unico, with a few limitations:

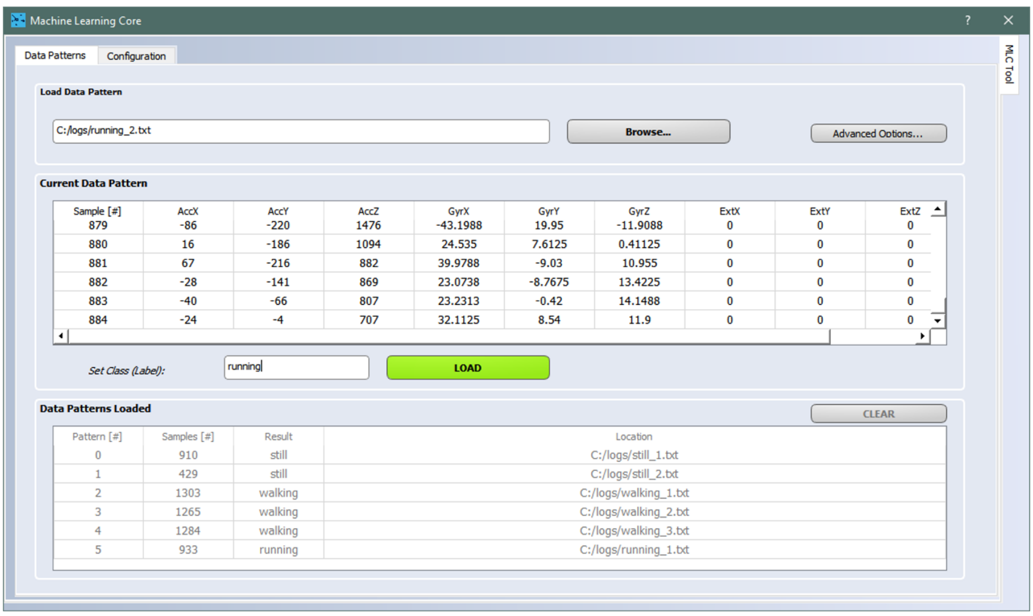
- Every data pattern has to start with a header line, containing the unit of measurement of the data
  - A_X [m*g*] A_Y [m*g*] A_Z [m*g*] G_X [dps] G_Y [dps] G_Z [dps]
- The data after the header line must be separated by “tab” or “space”.
- The order of sensors in the file columns must be accelerometer data (if available), gyroscope data (if available).
- The order of the axes in the columns of any sensor is X, Y, Z.

Figure 27. Configuration procedure



Opening the machine learning core tool available in Unico, the data patterns, acquired in the format described above, can be loaded assigning the expected result for each data log (as shown in the following figure).

Figure 28. Assigning a result to a data pattern



When all the data patterns have been loaded, the machine learning core parameters can be configured through the [**Configuration**] tab. These parameters are ODR, full scales, number of decision trees, window length, filters, features, and so on (as shown in Figure 29, Figure 30, Figure 31, Figure 32).

Figure 29. Configuration of machine learning core

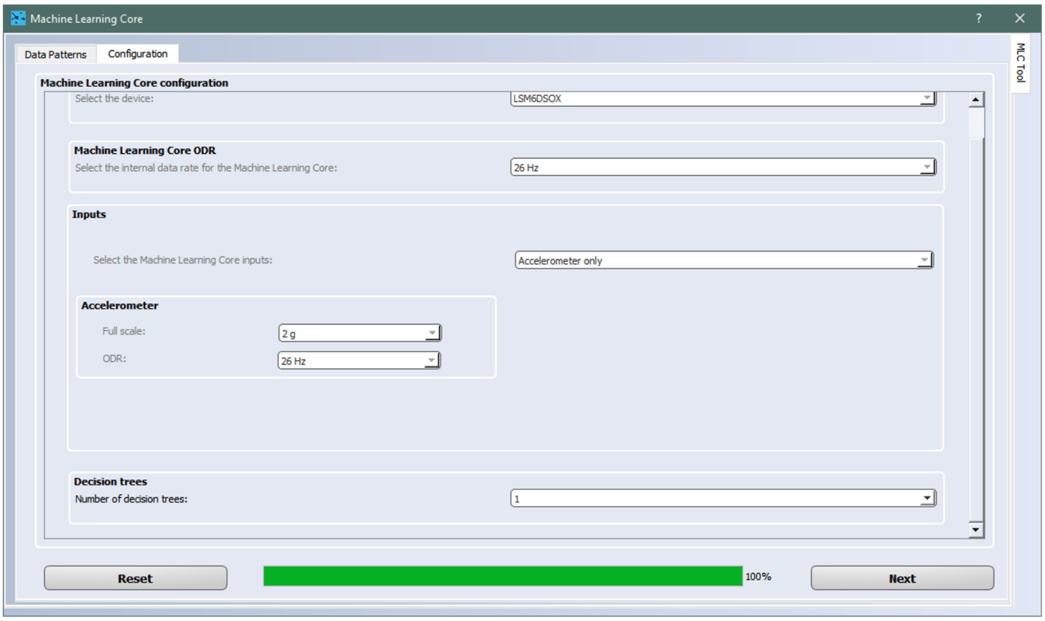


Figure 30. Configuration of filters

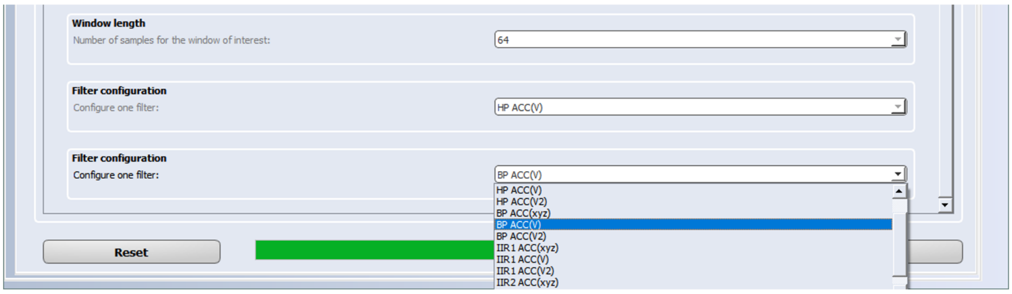


Figure 31. Configuration of features

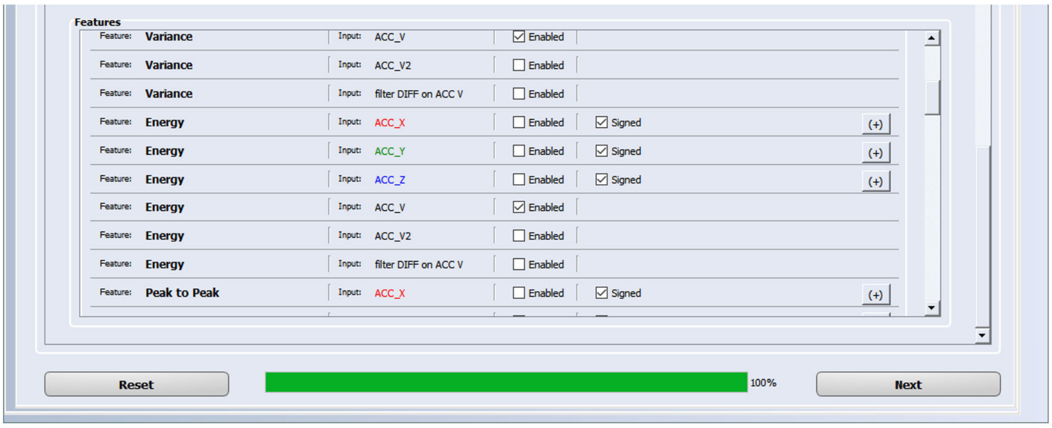
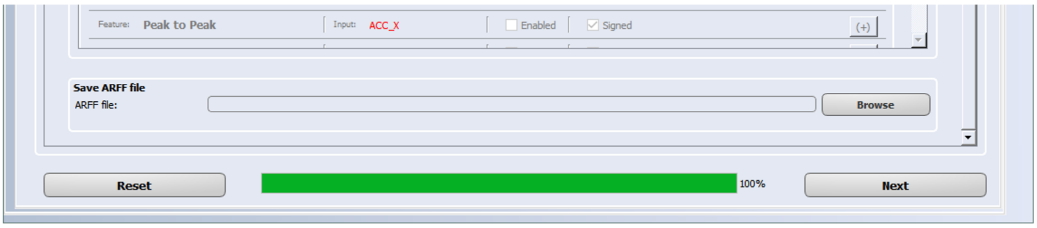


Figure 32. **ARFF generation**



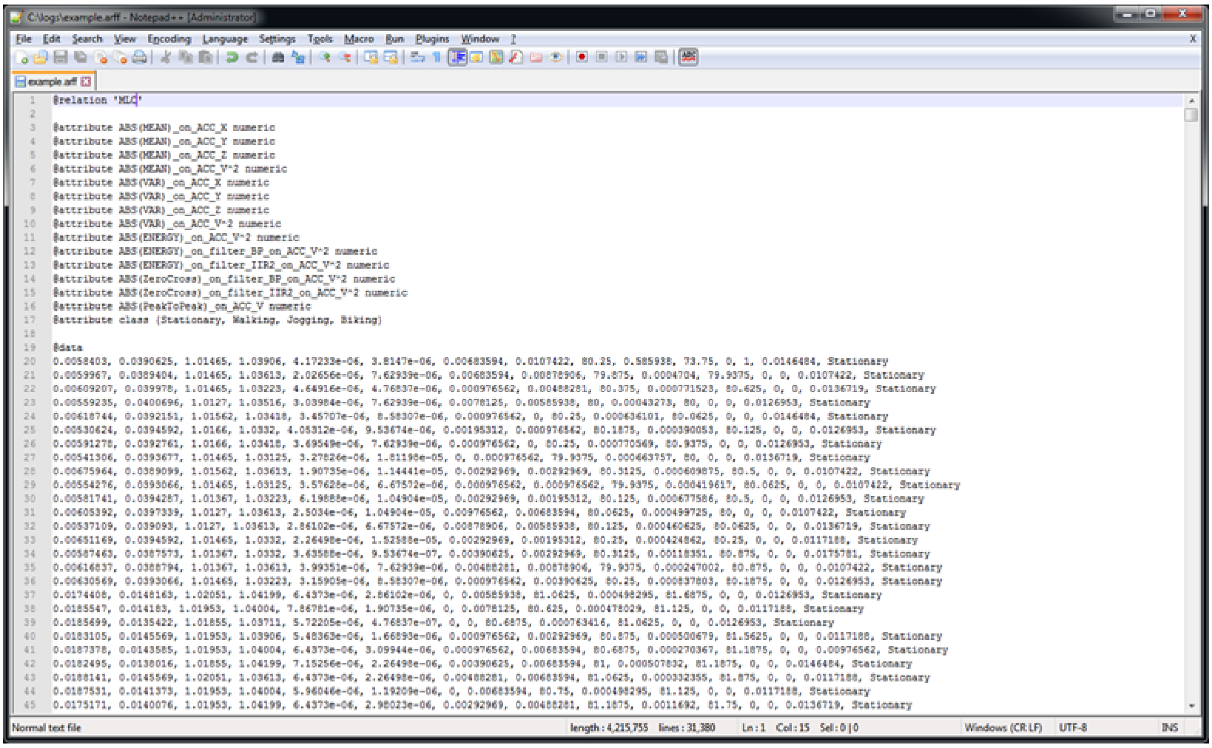
Multiple filters and multiple features can be chosen. The GUI iteratively asks for another filter until the parameter [**End filter configuration**] is chosen (Figure 30). All the available features can be easily selected using the checkboxes (Figure 31).

Once all the features have been configured, the machine learning core tool in Unico generates an ARFF file (Figure 32), which is the file containing all the features computed from the training data. Figure 33 shows an example of an ARFF file generated by the machine learning core tool in Unico.

Unico has a built-in tool for decision tree generation which internally uses the ARFF file to generate a decision tree, however, the ARFF file can also be used in external tools (for instance Weka). The decision tree generated with the external tool can be imported in Unico.

In some cases, the user must adapt the ARFF file to the file format required by other external tools for decision tree generation, and also the decision tree must be compatible with the format described in Section 2.2 Decision tree generation. For more information about the external tools supported, see the Appendix sections.

Figure 33. **ARFF file**



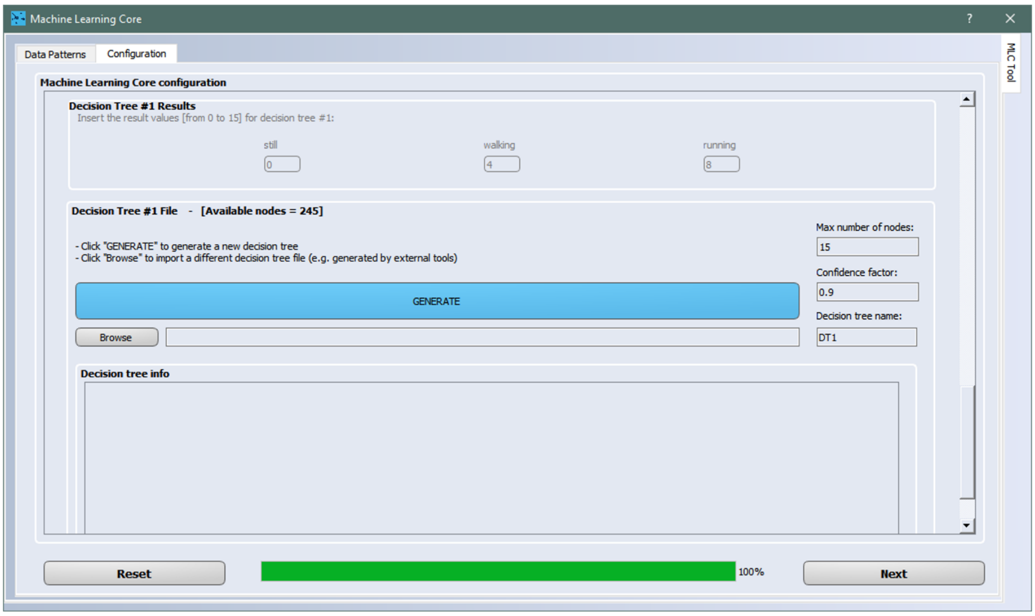

```
@relation 'MLC'

@attribute ABS(MEAN)_on_ACC_X numeric
@attribute ABS(MEAN)_on_ACC_Y numeric
@attribute ABS(MEAN)_on_ACC_Z numeric
@attribute ABS(MEAN)_on_ACC_V^2 numeric
@attribute ABS(VAR)_on_ACC_X numeric
@attribute ABS(VAR)_on_ACC_Y numeric
@attribute ABS(VAR)_on_ACC_Z numeric
@attribute ABS(VAR)_on_ACC_V^2 numeric
@attribute ABS(ENERGY)_on_ACC_V^2 numeric
@attribute ABS(ENERGY)_on_filter_BP_on_ACC_V^2 numeric
@attribute ABS(ENERGY)_on_filter_IIR2_on_ACC_V^2 numeric
@attribute ABS(ZeroCross)_on_filter_BP_on_ACC_V^2 numeric
@attribute ABS(ZeroCross)_on_filter_IIR2_on_ACC_V^2 numeric
@attribute ABS(PeakToPeak)_on_ACC_V numeric
@attribute class {Stationary, Walking, Jogging, Biking}

@data
0.0058403, 0.0390625, 1.01465, 1.03906, 4.17233e-06, 3.8147e-06, 0.00683594, 0.0107422, 80.25, 0.585938, 73.75, 0, 1, 0.0146484, Stationary
0.0059967, 0.0389404, 1.01465, 1.03613, 2.02656e-06, 7.62939e-06, 0.00683594, 0.00878906, 79.875, 0.0004704, 79.9375, 0, 0, 0.0107422, Stationary
0.00609207, 0.039978, 1.01465, 1.03223, 4.64916e-06, 4.76837e-06, 0.000976562, 0.00488281, 80.375, 0.000771523, 80.625, 0, 0, 0.0136719, Stationary
0.00559235, 0.0400696, 1.0127, 1.03516, 3.03984e-06, 7.62939e-06, 0.0078125, 0.00585938, 80, 0.00043273, 80, 0, 0, 0.0126953, Stationary
0.00618744, 0.0392151, 1.01562, 1.03418, 3.45707e-06, 8.58307e-06, 0.000976562, 0, 80.25, 0.000634101, 80.0625, 0, 0, 0.0146484, Stationary
0.00530624, 0.0394592, 1.0166, 1.0332, 4.05312e-06, 9.53674e-06, 0.00195312, 0.000976562, 80.1875, 0.000390053, 80.125, 0, 0, 0.0126953, Stationary
0.00591278, 0.0392761, 1.0166, 1.03418, 3.69549e-06, 7.62939e-06, 0.000976562, 0, 80.25, 0.000770569, 80.9375, 0, 0, 0.0126953, Stationary
0.00541306, 0.0393677, 1.01465, 1.03125, 3.27826e-06, 1.81198e-05, 0, 0.000976562, 79.9375, 0.000463757, 80, 0, 0, 0.0136719, Stationary
0.00675964, 0.0389099, 1.01562, 1.03613, 1.90735e-06, 1.14441e-05, 0.00292969, 0.00292969, 80.3125, 0.000609875, 80.5, 0, 0, 0.0107422, Stationary
0.00554276, 0.0393066, 1.01465, 1.03125, 3.57628e-06, 6.67572e-06, 0.000976562, 0.000976562, 79.9375, 0.000419617, 80.0625, 0, 0, 0.0107422, Stationary
0.00581741, 0.0394287, 1.01367, 1.03223, 6.19888e-06, 1.04904e-05, 0.00292969, 0.00195312, 80.125, 0.000677586, 80.5, 0, 0, 0.0126953, Stationary
0.00605392, 0.0397339, 1.0127, 1.03613, 2.5034e-06, 1.04904e-05, 0.00976562, 0.00683594, 80.0625, 0.000499725, 80, 0, 0, 0.0107422, Stationary
0.00537109, 0.039093, 1.0127, 1.03613, 2.86102e-06, 6.67572e-06, 0.00878906, 0.00585938, 80.125, 0.000460625, 80.0625, 0, 0, 0.0136719, Stationary
0.00651169, 0.0394592, 1.01465, 1.0332, 2.26498e-06, 1.52588e-05, 0.00292969, 0.00195312, 80.25, 0.000424862, 80.25, 0, 0, 0.0117188, Stationary
0.00587463, 0.0387573, 1.01367, 1.0332, 3.63588e-06, 9.53674e-07, 0.00390625, 0.00292969, 80.3125, 0.00118351, 80.875, 0, 0, 0.0175781, Stationary
0.00616837, 0.0388794, 1.01367, 1.03613, 3.99351e-06, 7.62939e-06, 0.00488281, 0.00878906, 79.9375, 0.000247002, 80.875, 0, 0, 0.0107422, Stationary
0.00630569, 0.0393066, 1.01465, 1.03223, 3.15905e-06, 8.58307e-06, 0.000976562, 0.00390625, 80.25, 0.000837803, 80.1875, 0, 0, 0.0126953, Stationary
0.0174408, 0.0148163, 1.02051, 1.04199, 6.4373e-06, 2.86102e-06, 0, 0.00585938, 81.0625, 0.000498295, 81.6875, 0, 0, 0.0126953, Stationary
0.0185547, 0.014183, 1.01953, 1.04004, 7.86781e-06, 1.90735e-06, 0, 0.0078125, 80.625, 0.000478029, 81.125, 0, 0, 0.0117188, Stationary
0.0185699, 0.0135422, 1.01855, 1.03711, 5.72205e-06, 4.76837e-07, 0, 0, 80.6875, 0.000763416, 81.0625, 0, 0, 0.0126953, Stationary
0.0183105, 0.0145569, 1.01953, 1.03906, 5.48363e-06, 1.66893e-06, 0.000976562, 0.00292969, 80.875, 0.000500679, 81.5625, 0, 0, 0.0117188, Stationary
0.0187378, 0.0143585, 1.01953, 1.04004, 6.4373e-06, 3.09944e-06, 0.000976562, 0.00683594, 80.6875, 0.000270367, 81.1875, 0, 0, 0.00976562, Stationary
0.0182495, 0.0138016, 1.01855, 1.04199, 7.15256e-06, 2.26498e-06, 0.00390625, 0.00683594, 81, 0.000507832, 81.1875, 0, 0, 0.0146484, Stationary
0.0188141, 0.0145569, 1.02051, 1.03613, 6.4373e-06, 2.26498e-06, 0.00488281, 0.00683594, 81.0625, 0.000332355, 81.875, 0, 0, 0.0117188, Stationary
0.0187531, 0.0141373, 1.01953, 1.04004, 5.96046e-06, 1.19209e-06, 0, 0.00683594, 80.75, 0.000498295, 81.125, 0, 0, 0.0117188, Stationary
0.0175171, 0.0140076, 1.01953, 1.04199, 6.4373e-06, 2.98023e-06, 0.00292969, 0.00488281, 81.1875, 0.0011692, 81.75, 0, 0, 0.0136719, Stationary
```

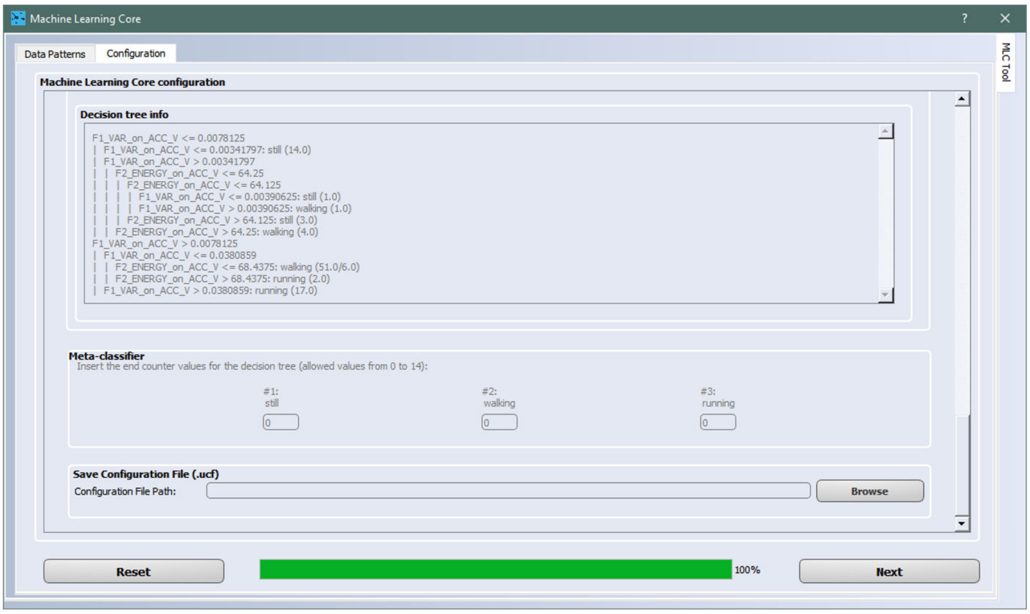

Before generating or loading the decision tree, Unico also asks for the result values associated to each class recognized by the decision tree. These values are used as possible values for the MLCx_SRC registers.

Figure 34. **Configuration of results and decision tree**



The last step of the configuration process is to configure the [**Meta-classifier**], which is the optional filter for the generation of the decision tree results. After that, the tool is ready to generate a configuration for the device (Figure 35).

Figure 35. **Meta-classifier and device configuration**



Once the MLC configuration has been completed, Unico allows loading the .ucf file generated to directly program the device. The loading feature is available in the [**Load/Save**] tab of the Unico main window (Figure 37). Alternatively, at the end of the MLC configuration a checkbox allows directly loading the configuration created on the device as shown in Figure 36.

Figure 36. Creation of configuration file

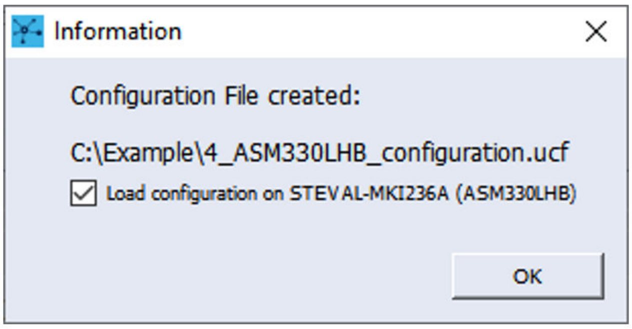


Figure 37. Unico load configuration

When the device is programmed, the machine learning core results can be monitored in the [**Data**] window of Unico (Figure 38) or in one of the registers tabs containing the machine learning core source registers (Figure 39).

The .ucf file generated by Unico can also be used for integrating the generated MLC configuration in other platforms and software (for example, AlgoBuilder, Unicleo, SensorTile.box, and so forth).

From the .ucf file it is also possible to convert the sequence of values to a header file (.h) to be imported in any C project (for example, driver, firmware, and so on): Unico allows .h file generation (from .ucf files) through the "C code generation" dedicated tool in the options tab of the Unico main window. An example of using the generated .h file in a standard C driver is available in [STMems_Standard_C_drivers repository] on GitHub.

Figure 38. Unico data window

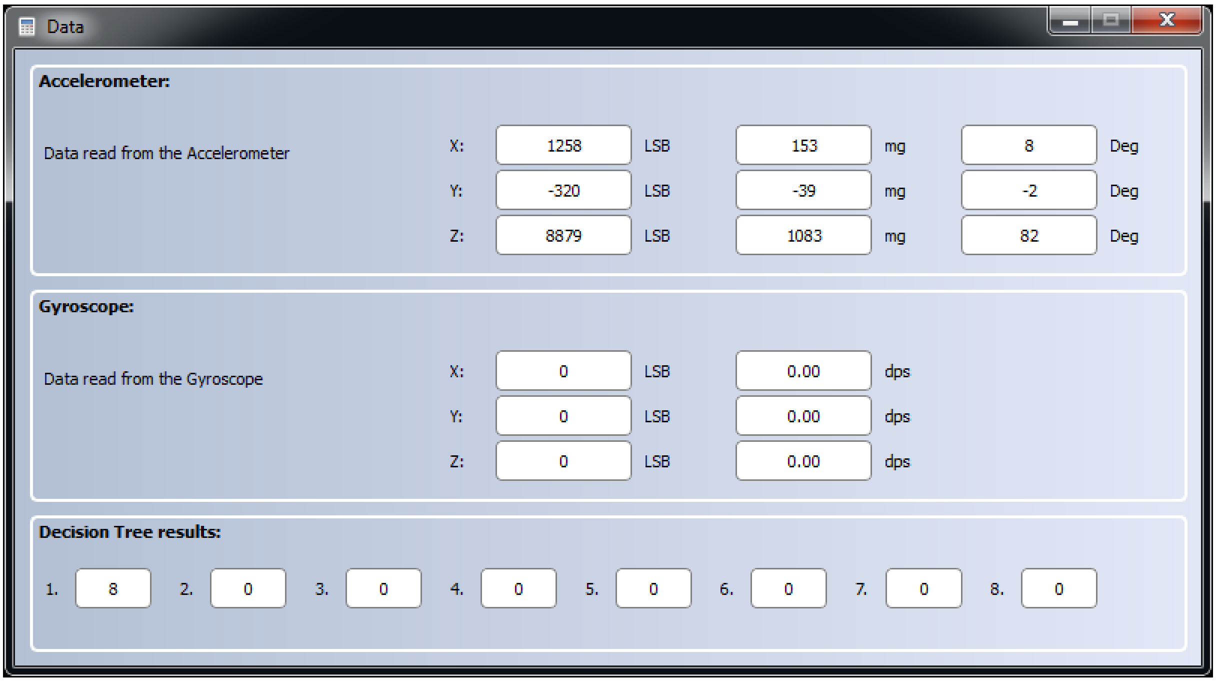
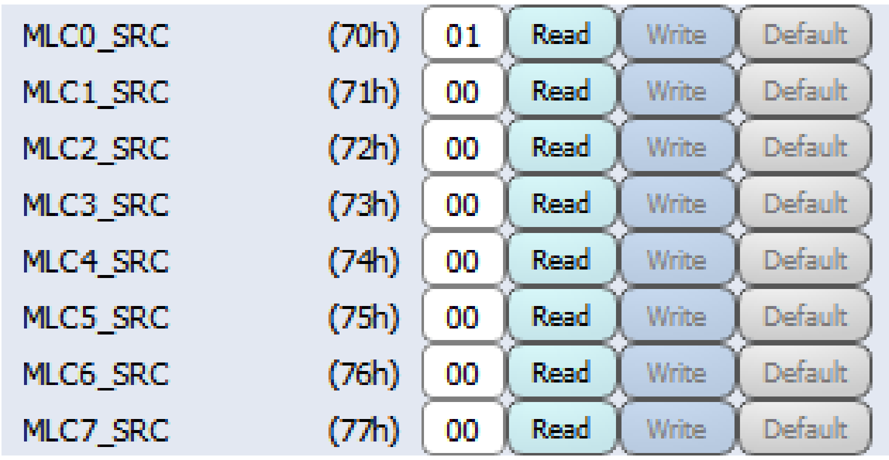


Figure 39. Unico - machine learning core source registers



Appendix A Weka

Weka is free software developed at the University of Waikato, New Zealand. It cointains a collection of visualization tools and algorithms for data analysis and predictive modeling, together with graphical user interfaces for easy access to these functions.

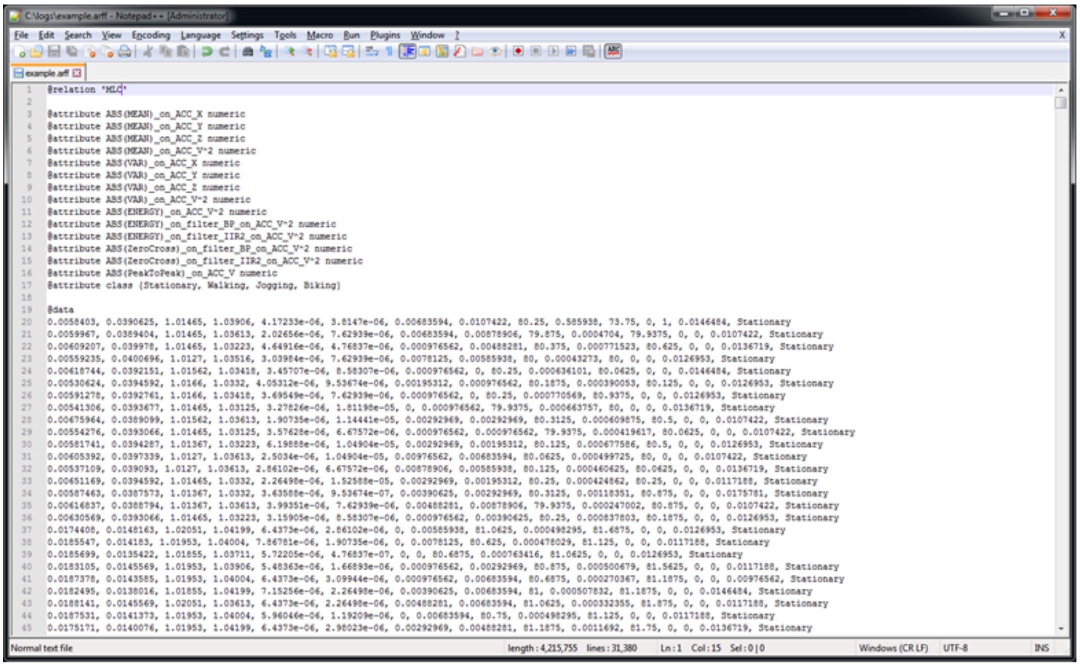
Weka is one of the most popular machine learning tools for decision tree generation. This section contains some details about this external software, additional details can be found at the links below:

- Weka download
- Weka website
- Weka user guide

All of Weka's techniques are predicated on the assumption that the data is available as one flat file or relation, where each data point is described by a fixed number of attributes.

An ARFF (attribute-relation file format) file is an ASCII text file that describes a list of instances sharing a set of attributes. The ARFF files have two distinct sections, as shown in Figure 40: a header section containing the attributes (features, classes), and a data section containing all the feature values together with the corresponding class to be associated to that set of features.

Figure 40. ARFF example




```
@relation "MLC"

@attribute ABS(MEAN)_on_ACC_X numeric
@attribute ABS(MEAN)_on_ACC_Y numeric
@attribute ABS(MEAN)_on_ACC_Z numeric
@attribute ABS(MEAN)_on_ACC_V^2 numeric
@attribute ABS(VAR)_on_ACC_X numeric
@attribute ABS(VAR)_on_ACC_Y numeric
@attribute ABS(VAR)_on_ACC_Z numeric
@attribute ABS(VAR)_on_ACC_V^2 numeric
@attribute ABS(ENERGY)_on_ACC_V^2 numeric
@attribute ABS(ENERGY)_on_filter_BP_on_ACC_V^2 numeric
@attribute ABS(ENERGY)_on_filter_IIR2_on_ACC_V^2 numeric
@attribute ABS(ZeroCross)_on_filter_BP_on_ACC_V^2 numeric
@attribute ABS(ZeroCross)_on_filter_IIR2_on_ACC_V^2 numeric
@attribute ABS(PeakToPeak)_on_ACC_V numeric
@attribute class {Stationary, Walking, Jogging, Biking}

@data
0.0058403, 0.0390625, 1.01465, 1.03906, 4.17233e-06, 3.8147e-06, 0.00683594, 0.0107422, 80.25, 0.585938, 73.75, 0, 1, 0.0146484, Stationary
0.0059967, 0.0389404, 1.01465, 1.03613, 2.02656e-06, 7.62939e-06, 0.00683594, 0.00878906, 79.875, 0.0004704, 79.9375, 0, 0, 0.0107422, Stationary
0.00609207, 0.039978, 1.01465, 1.03223, 4.64916e-06, 4.76837e-06, 0.000976562, 0.00488281, 80.375, 0.000771523, 80.625, 0, 0, 0.0136719, Stationary
0.00559235, 0.0400696, 1.0127, 1.03516, 3.03984e-06, 7.62939e-06, 0.0078125, 0.00585938, 80, 0.00043273, 80, 0, 0, 0.0126953, Stationary
0.00618744, 0.0392151, 1.01562, 1.03418, 3.45707e-06, 8.58307e-06, 0.000976562, 0, 80.25, 0.000636101, 80.0625, 0, 0, 0.0146484, Stationary
0.00530624, 0.0394592, 1.0166, 1.0332, 4.05312e-06, 9.53674e-06, 0.00195312, 0.000976562, 80.1875, 0.000390053, 80.125, 0, 0, 0.0126953, Stationary
0.00591278, 0.0392761, 1.0166, 1.03418, 3.69549e-06, 7.62939e-06, 0.000976562, 0, 80.25, 0.000770569, 80.9375, 0, 0, 0.0126953, Stationary
0.00541306, 0.0393677, 1.01465, 1.03125, 3.27826e-06, 1.81198e-05, 0, 0.000976562, 79.9375, 0.000663757, 80, 0, 0, 0.0136719, Stationary
0.00675964, 0.0389099, 1.01562, 1.03613, 1.90735e-06, 1.14441e-05, 0.00292969, 0.00292969, 80.3125, 0.000609875, 80.5, 0, 0, 0.0107422, Stationary
0.00554276, 0.0393066, 1.01465, 1.03125, 3.57628e-06, 6.67572e-06, 0.000976562, 0.000976562, 79.9375, 0.000419617, 80.0625, 0, 0, 0.0107422, Stationary
0.00581741, 0.0394287, 1.01367, 1.03223, 6.19888e-06, 1.04904e-05, 0.00292969, 0.00195312, 80.125, 0.000677586, 80.5, 0, 0, 0.0126953, Stationary
0.00605392, 0.0397339, 1.0127, 1.03613, 2.5034e-06, 1.04904e-05, 0.00976562, 0.00683594, 80.0625, 0.000499725, 80, 0, 0, 0.0107422, Stationary
0.00537109, 0.039093, 1.0127, 1.03613, 2.86102e-06, 6.67572e-06, 0.00878906, 0.00585938, 80.125, 0.000460625, 80.0625, 0, 0, 0.0136719, Stationary
0.0065169, 0.0394592, 1.01465, 1.0332, 2.26498e-06, 1.52588e-05, 0.00292969, 0.00195312, 80.25, 0.000424862, 80.25, 0, 0, 0.0117188, Stationary
0.00587463, 0.0387573, 1.01367, 1.0332, 3.63588e-06, 9.53674e-07, 0.00390625, 0.00292969, 80.3125, 0.00118351, 80.875, 0, 0, 0.0175781, Stationary
0.00616837, 0.0388794, 1.01367, 1.03613, 3.99351e-06, 7.62939e-06, 0.00488281, 0.00878906, 79.9375, 0.000247002, 80.875, 0, 0, 0.0107422, Stationary
0.00630569, 0.0393066, 1.01465, 1.03223, 3.15905e-06, 8.58307e-06, 0.000976562, 0.00390625, 80.25, 0.000837803, 80.1875, 0, 0, 0.0126953, Stationary
0.0174408, 0.0148163, 1.02051, 1.04199, 6.4373e-06, 2.86102e-06, 0, 0.00585938, 81.0625, 0.000498295, 81.6875, 0, 0, 0.0126953, Stationary
0.0185547, 0.014183, 1.01953, 1.04004, 7.86781e-06, 1.90735e-06, 0, 0.0078125, 80.625, 0.000478029, 81.125, 0, 0, 0.0117188, Stationary
0.0185699, 0.0135422, 1.01855, 1.03711, 5.72205e-06, 4.76837e-07, 0, 0, 80.6875, 0.000763416, 81.0625, 0, 0, 0.0126953, Stationary
0.0183105, 0.0145569, 1.01953, 1.03906, 5.48363e-06, 1.66893e-06, 0.000976562, 0.00292969, 80.875, 0.000500679, 81.5625, 0, 0, 0.0117188, Stationary
0.0187378, 0.0143585, 1.01953, 1.04004, 6.4373e-06, 3.09944e-06, 0.000976562, 0.00683594, 80.6875, 0.000270367, 81.1875, 0, 0, 0.00976562, Stationary
0.0182495, 0.0138016, 1.01855, 1.04199, 7.15256e-06, 2.26498e-06, 0.00390625, 0.00683594, 81, 0.000507832, 81.1875, 0, 0, 0.0146484, Stationary
0.0188141, 0.0145569, 1.02051, 1.03613, 6.4373e-06, 2.26498e-06, 0.00488281, 0.00683594, 81.0625, 0.000332355, 81.875, 0, 0, 0.0117188, Stationary
0.0187531, 0.0141373, 1.01953, 1.04004, 5.96046e-06, 1.19209e-06, 0, 0.00683594, 80.75, 0.000498295, 81.125, 0, 0, 0.0117188, Stationary
0.0175171, 0.0140076, 1.01953, 1.04199, 6.4373e-06, 2.98023e-06, 0.00292969, 0.00488281, 81.1875, 0.0011692, 81.75, 0, 0, 0.0136719, Stationary
```

Figure 41. Weka GUI Chooser

When launching Weka, the [**Weka GUI Chooser**] window appears (Figure 41), and the [**Explorer**] section, selectable through the first button, is the Weka main user interface.

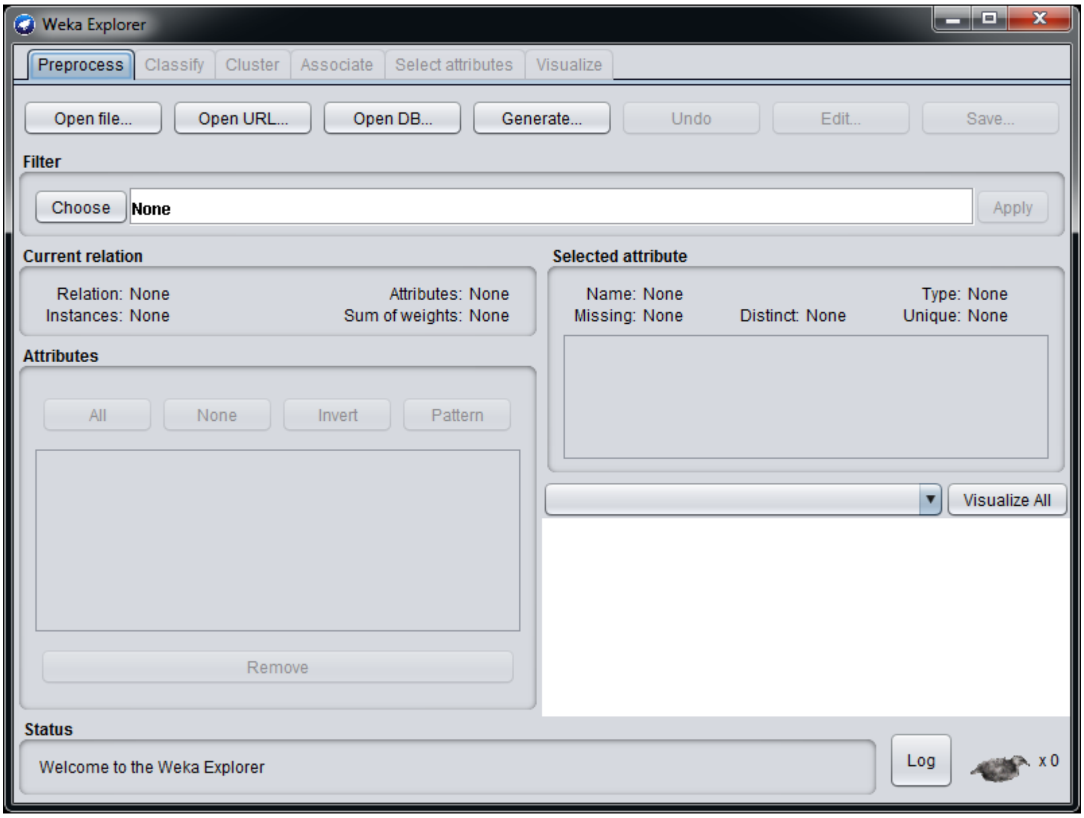
When selecting the Weka [**Explorer**] a new interface appears (Figure 42). Several tabs are available in the [**Explorer**] interface:

- The [**Preprocess**] tab has facilities for importing data.
- The [**Classify**] tab allows applying classification and regression algorithms to the dataset in order to estimate accuracy of the resulting predictive model and to visualize erroneous predictions.
- The [**Cluster**] tab gives access to the clustering techniques in Weka.
- The [**Associate**] tab provides access to association rule learners that attempt to identify all important interrelationships between attributes in the data.
- The [**Select attributes**] tab provides algorithms for identifying the most predictive attributes in a dataset.
- The [**Visualize**] tab shows a scatter plot matrix.

In this appendix section, only the [**Preprocess**] and [**Classify**] tabs are described.

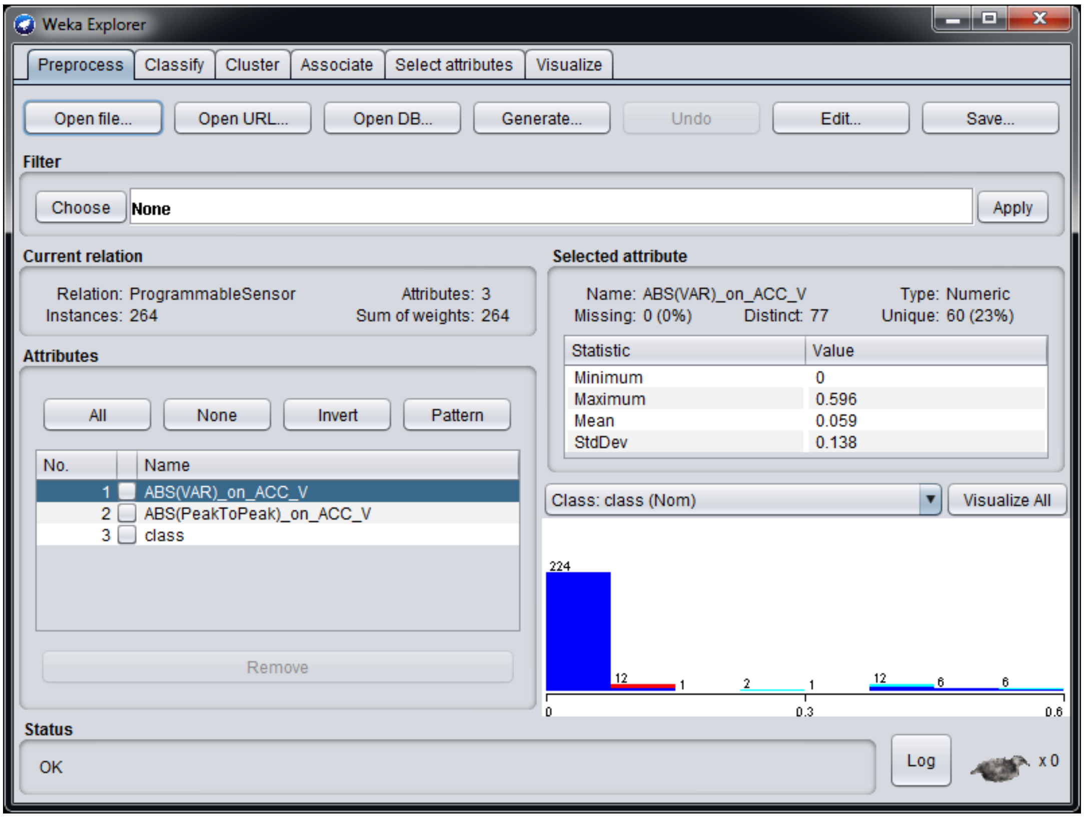
The [**Preprocess**] tab is shown in Figure 42, it allows loading an ARFF file from the [**Open file**] button.

Figure 42. **Weka Explorer**



When the ARFF file has been loaded, the [**Preprocess**] tab shows all the attributes (features and classes) of the imported ARFF file. The attributes can be visualized in a graphical way and the user can select the attributes to be used for the classification.

Figure 43. Weka Explorer - Attributes



After choosing the attributes, a classifier can be configured in the [**Classify**] tab of Weka [**Explorer**] (Figure 44). There are many classifiers available in Weka: by choosing the classifier [**J48**] (under [**trees**]) a decision tree can be generated (Figure 45).

Figure 44. Weka Classify

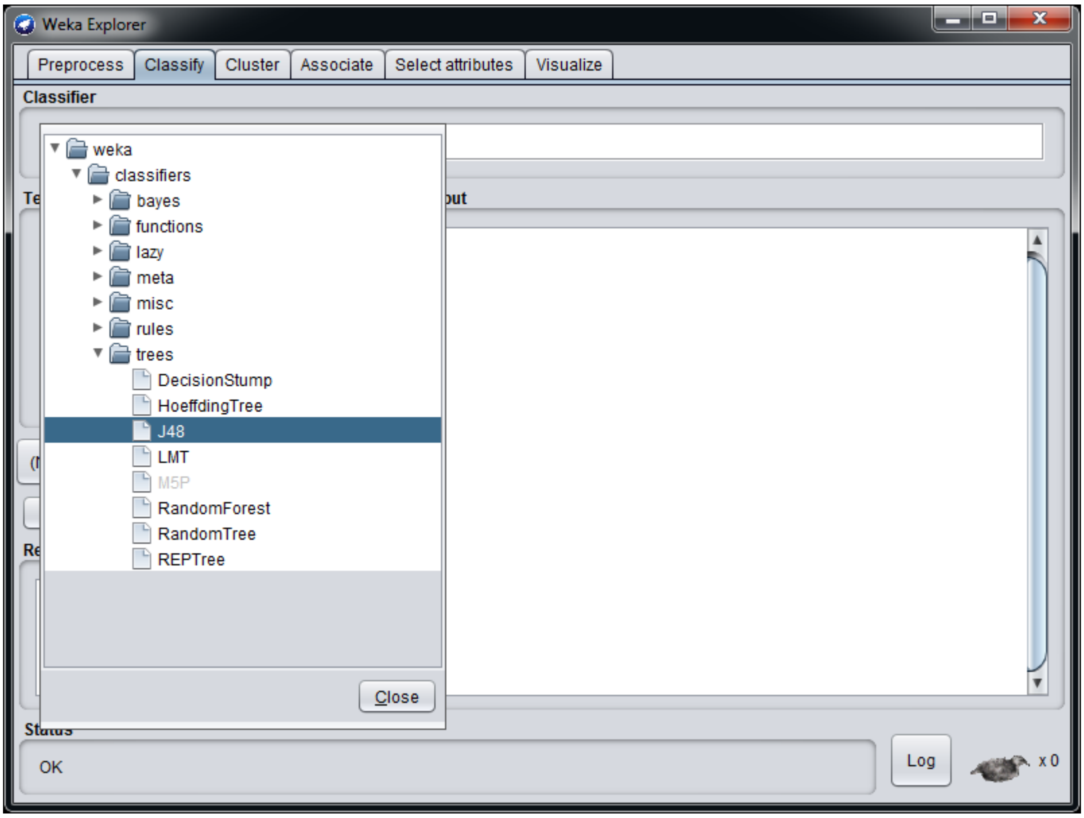
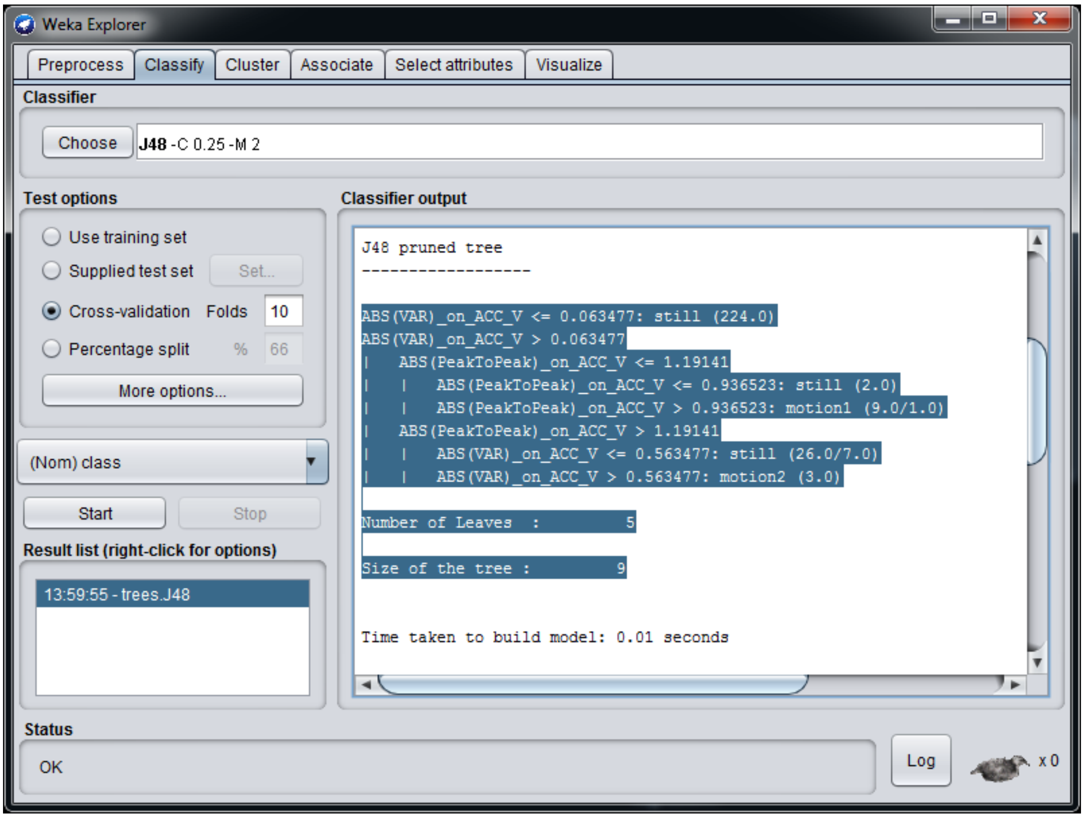
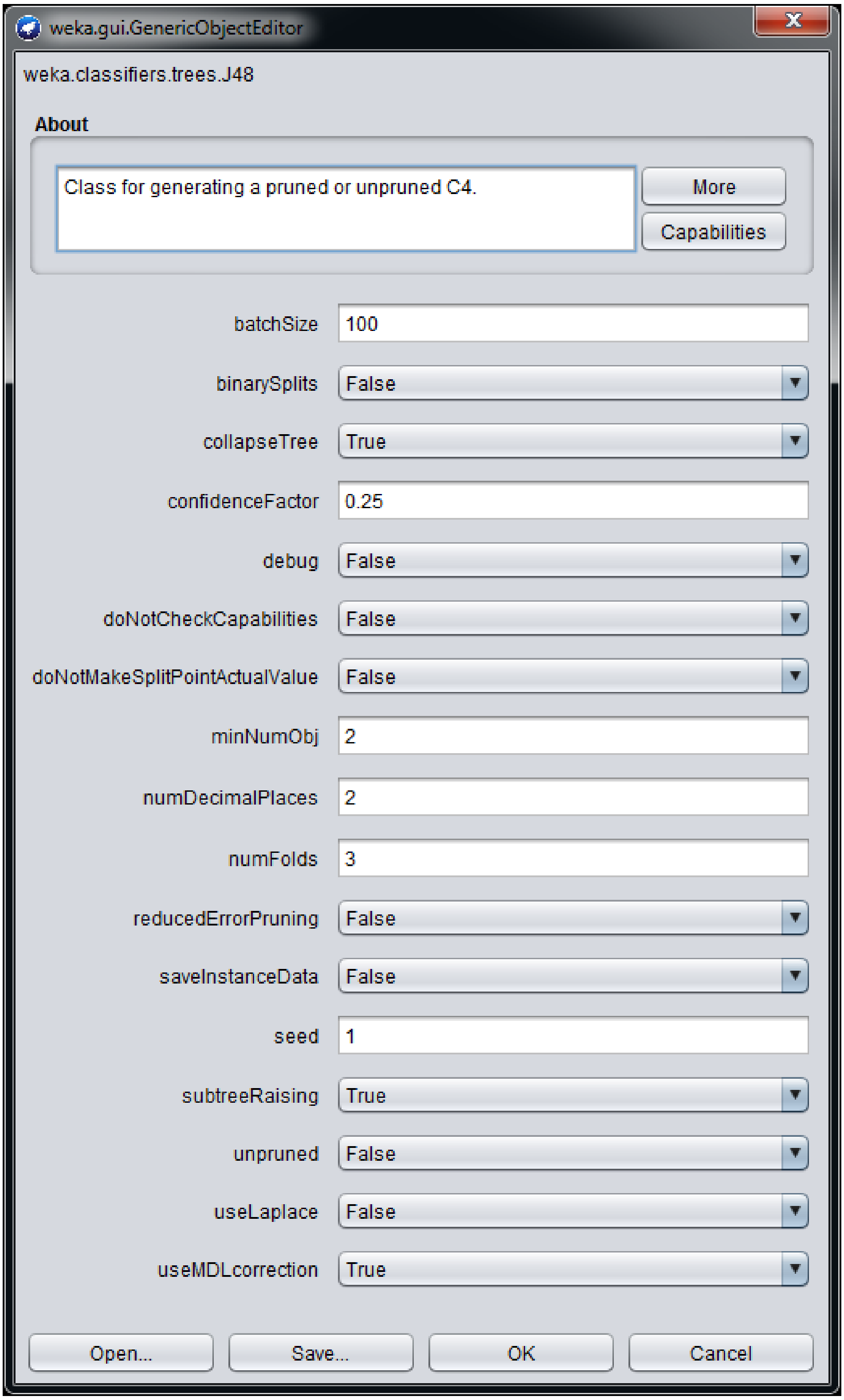


Figure 45. Weka Classify J48



Many parameters can be changed in the classifier section (Figure 46), and different decision trees can be generated by clicking the [**Start**] button (see Figure 45).

Figure 46. Weka J48 classifier parameters



All the decision trees generated can be easily compared in terms of:

- Number of nodes

Since the decision tree generated by the J48 algorithm in Weka is a binary tree, the number of nodes can be obtained by subtracting one from the parameter "Number of Leaves" which appears in the first row just after the decision tree (see Figure 47. Correctly classified instances). It is also possible to visualize the decision tree graphically by right-clicking on the [**Result list**] section on the left part of the tool (where all the models created can be easily compared).

- Correctly classified instances

It is an estimate of the accuracy of the model created. The result of the model is compared to the result provided by the labels. Figure 47. Correctly classified instances shows the correctly classified instances of an activity recognition model.

- Confusion matrix

An NxN table that summarizes how successful the classification model predictions were, that is, the correlation between the label and the model classification. One axis of a confusion matrix is the label that the model predicted, and the other axis is the actual label.

Figure 47. Correctly classified instances

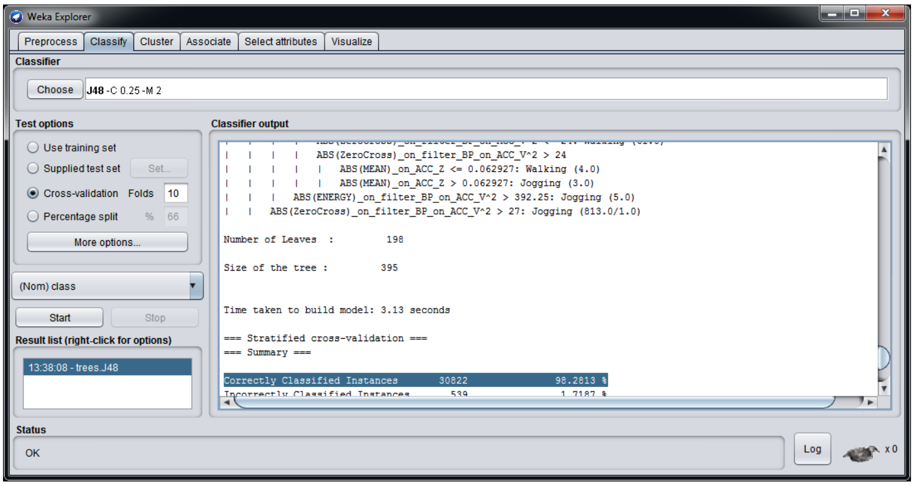
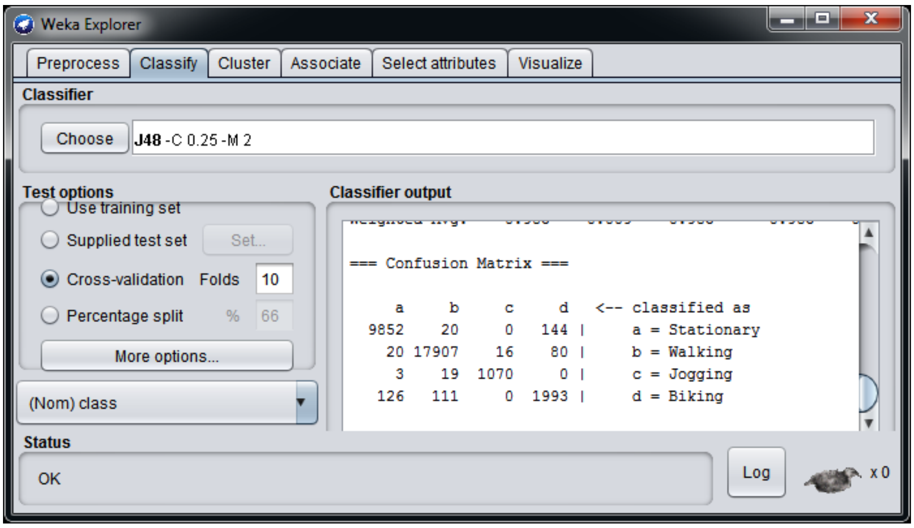


Figure 48. Confusion matrix shows an example of a confusion matrix for an activity recognition algorithm with four classes (stationary, walking, jogging, biking).

Figure 48. Confusion matrix



Appendix B RapidMiner

RapidMiner is a data science software platform which provides an integrated environment for data preparation, machine learning, deep learning, text mining, and predictive analytics. It is used for business and commercial applications as well as for research, education, training, rapid prototyping, and application development and supports all steps of the machine learning process including data preparation, results visualization, model validation and optimization.

This appendix describes the process to generate a decision tree starting from an ARFF file, using RapidMiner Studio. A simple example of a hand-washing detection algorithm is considered for this purpose. After opening RapidMiner Studio, the main steps are the following:

1. Add the [**Open File**] operator from the [**Operators**] window on the left, and drag the operator to the blank [**Process**] window as shown in Figure 49.
2. Double-click the [**Open File**] operator to choose the ARFF file to be loaded.
3. Find the [**Read ARFF**] operator and drag it to the [**Process**] window. Then connect the [**Read ARFF**] operator to the [**Open File**] operator as shown in Figure 50.
4. Find the [**Set Role**] operator and drag it to the [**Process**] window. Then, double-click the [**Set Role**] operator and type the attribute name and target role in the [**Parameters**] window as shown in Figure 51.
5. Find the [**Decision Tree**] operator and set the corresponding parameters as shown in Figure 52. You also need to connect the [**Decision Tree**] operator to [**res**].
6. Click the [**Run**] button (blue triangle icon) in the upper left section of RapidMiner Studio.
7. After the [**Run**] button has been clicked, the [**Results**] tab shows the decision tree generated, in terms of [**Graph**] (Figure 53) and [**Description**].
8. In the [**Description**] section of the decision tree generated (Figure 54) you need to copy the decision tree to a text file, which can be imported in the MLC tool in Unico.

Figure 49. **RapidMiner Studio - Open File**

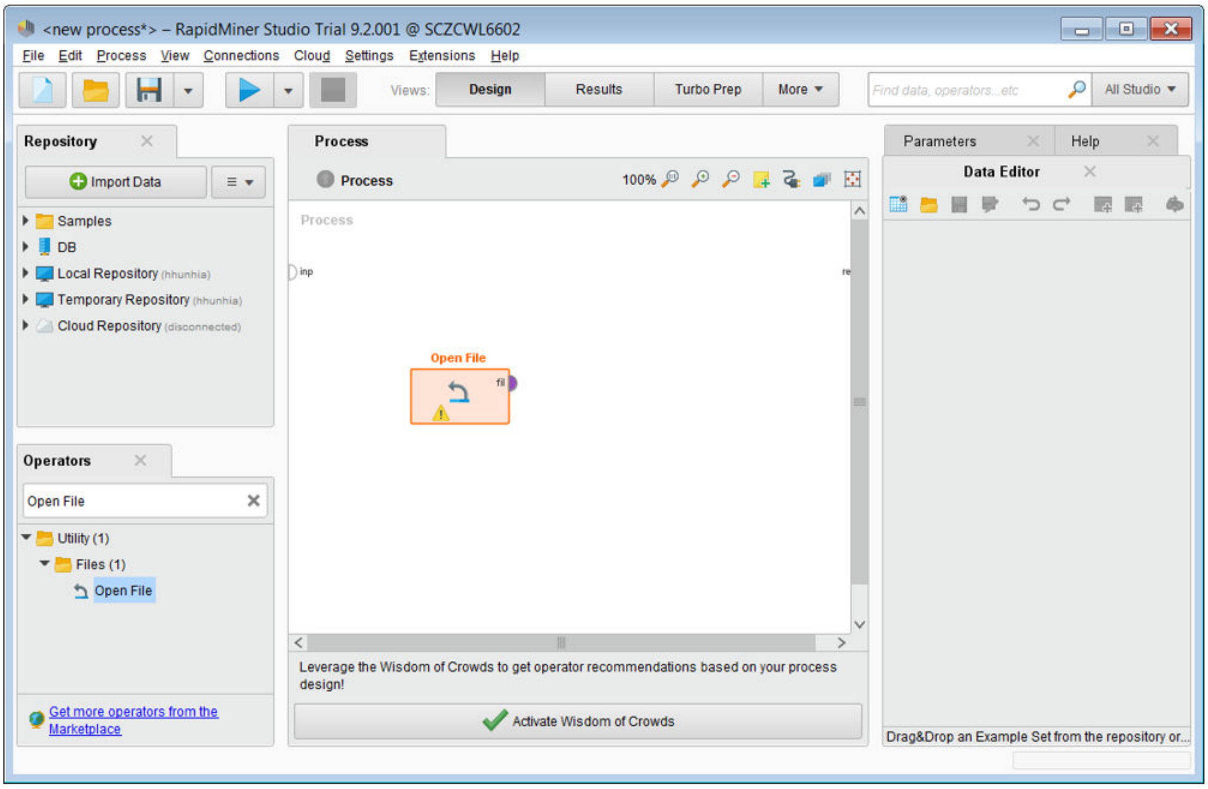


Figure 50. RapidMiner Studio - Read ARFF

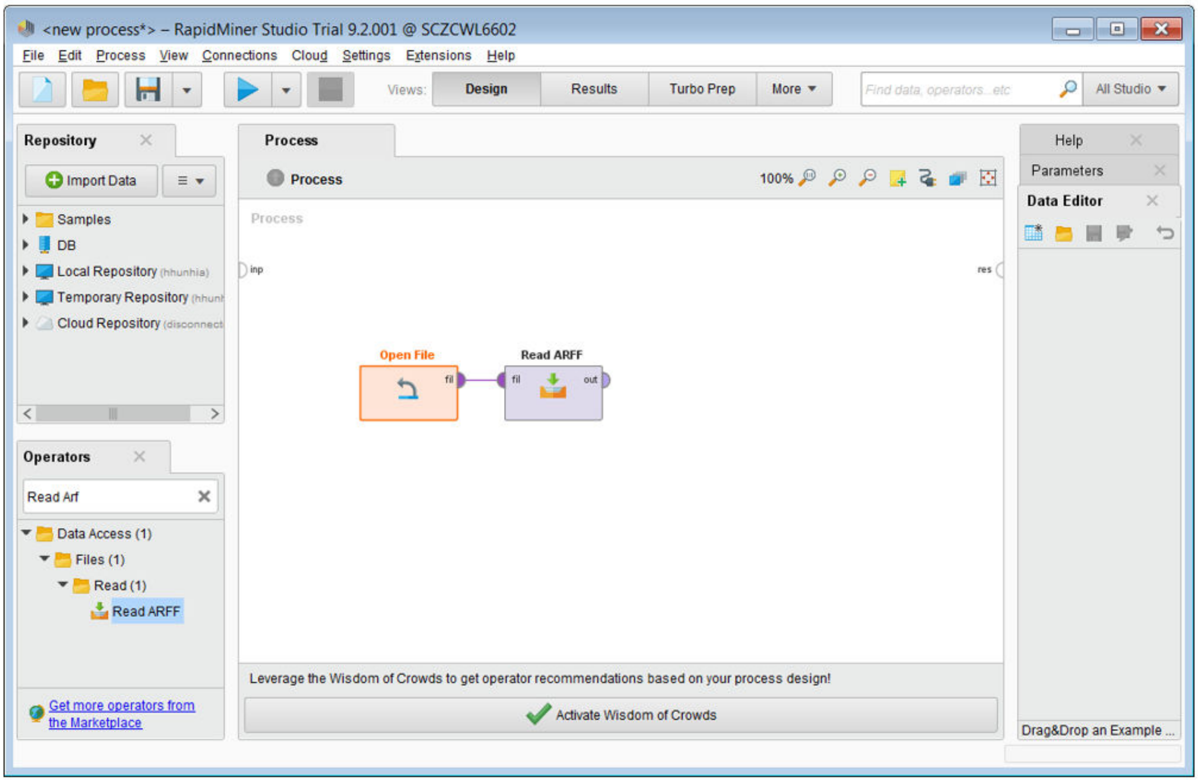


Figure 51. RapidMiner Studio - Set Role

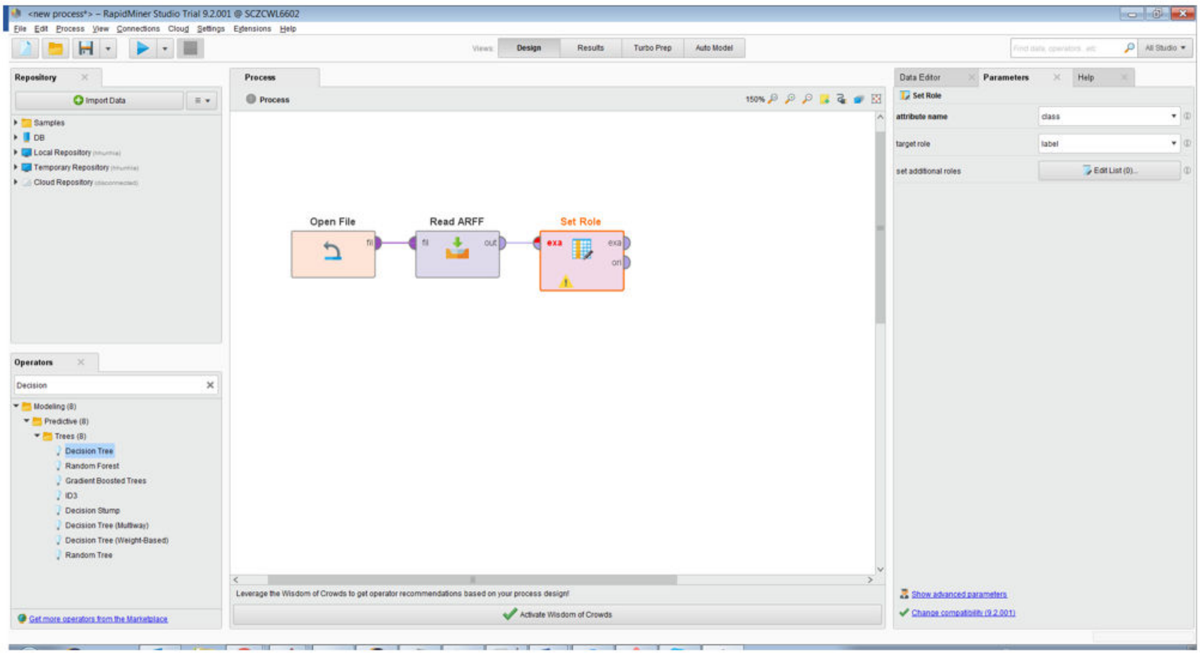


Figure 52. RapidMiner Studio - Decision Tree operator

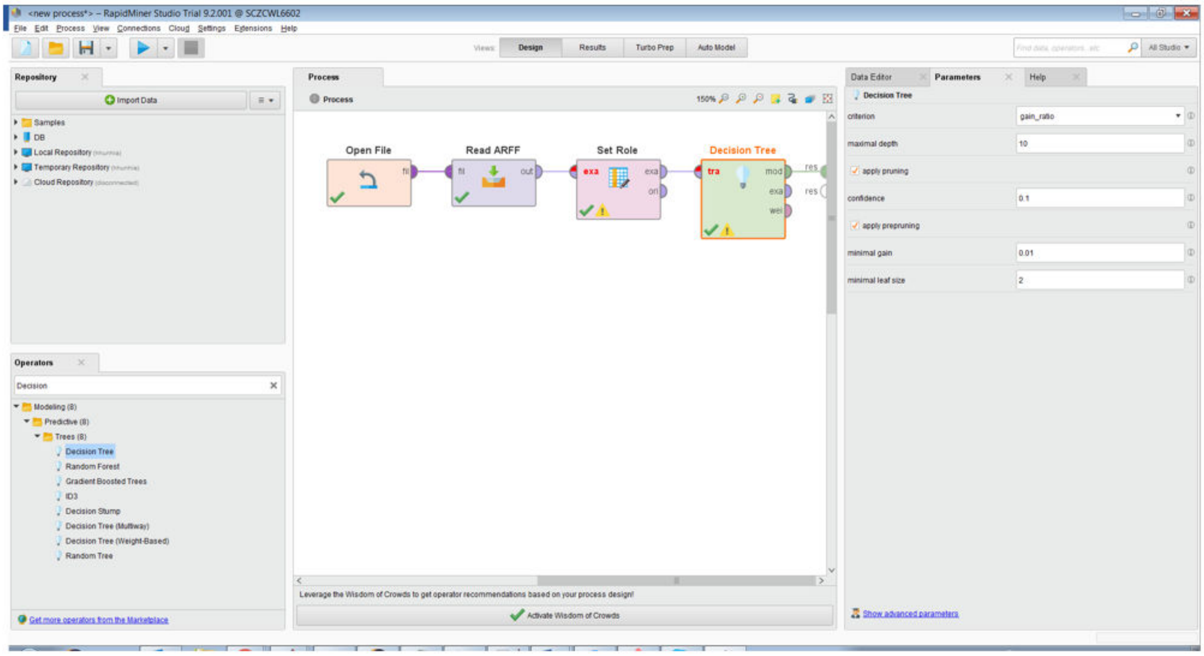


Figure 53. RapidMiner Studio - Decision Tree graph

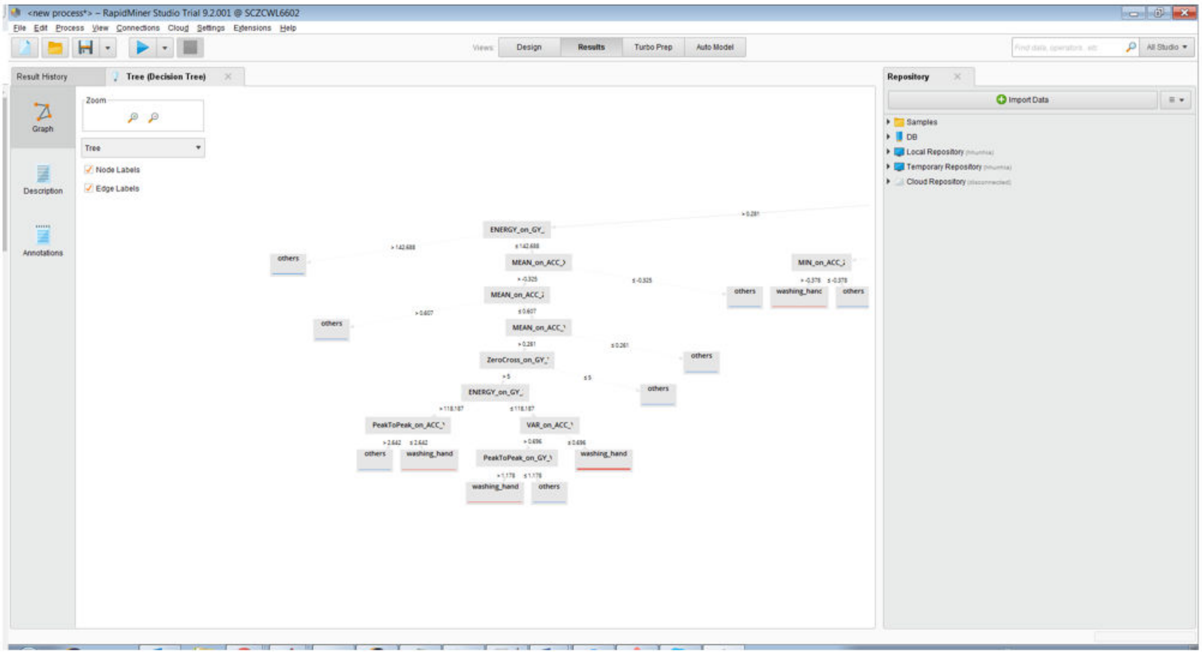
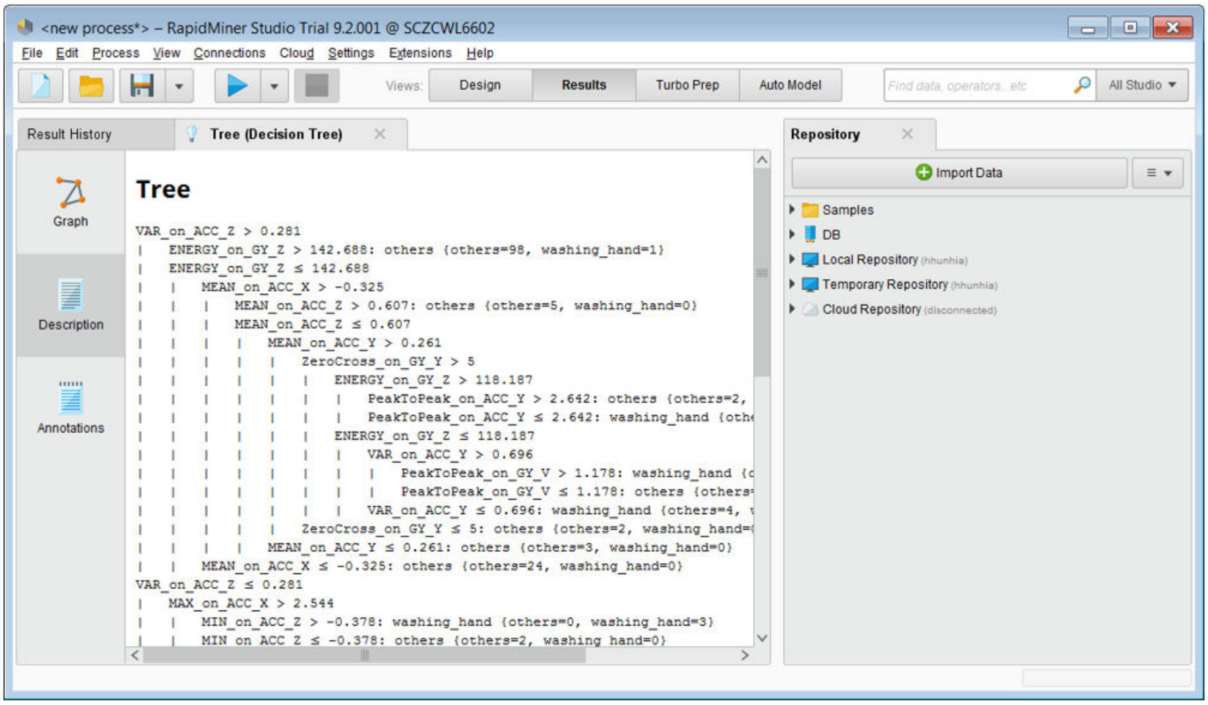


Figure 54. RapidMiner Studio - Decision Tree description



Appendix C Matlab

Decision trees for the machine learning core can be generated with Matlab. Dedicated scripts for Matlab are available at Matlab.

After importing all the scripts in the Matlab workspace, the function [**Generate_DecisionTree()**] should be called, specifying two file names (an *.arff* file containing the features computed by the machine learning core tool in Unico and a *.txt* file which contains the decision tree generated):

filename_ARFF = 'features.arff';

filename_dectree = 'decision_tree.txt';

Generate_DecisionTree(filename_ARFF, filename_dectree);

More details can be found in the *README.md* file available contained in the [**matlab**] folder of the GitHub repository.

Appendix D Python

Decision trees for the machine learning core can be generated with Python through the the “*scikit*” package.

Python scripts are available at Python both as a Jupyter notebook (*.ipynb) and as a common Python script (*.py). More details can be found in the *README.md* file available contained in the [**python**] folder of the GitHub repository.

Appendix E Glossary

This section contains a glossary of terms used in machine learning. Most of the terms have been taken from https://developers.google.com/machine-learning/glossary/.

ARFF	An ARFF (attribute-relation file format) file is an ASCII text file that describes a list of instances sharing a set of attributes. ARFF files were developed by the Machine Learning Project at the Department of Computer Science of The University of Waikato for use with the Weka machine learning software.
Attribute/feature	An attribute is an aspect of an instance (for example, temperature, humidity). Attributes are often called features in machine learning. A special attribute is the class label that defines the class this instance belongs to (required for supervised learning).
Binary classification	A type of classification task that outputs one of two mutually exclusive classes. For example, a machine learning model that evaluates email messages and outputs either "spam" or "not spam" is a binary classifier.
Class	One of a set of enumerated target values for a label. For example, in a binary classification model that detects spam, the two classes are spam and not spam. In a multi-class classification model that identifies dog breeds, the classes would be poodle, beagle, pug, and so on.
Classification model	A type of machine learning model for distinguishing among two or more discrete classes. For example, a natural language processing classification model could determine whether an input sentence was in French, Spanish, or Italian.
Classification threshold	A scalar-value criterion that is applied to a model's predicted score in order to separate the positive class from the negative class. Used when mapping logistic regression results to binary classification. For example, consider a logistic regression model that determines the probability of a given email message being spam. If the classification threshold is 0.9, then logistic regression values above 0.9 are classified as spam and those below 0.9 are classified as not spam.
Class-imbalanced dataset	A binary classification problem in which the labels for the two classes have significantly different frequencies. For example, a disease dataset in which 0.0001 of examples have positive labels and 0.9999 have negative labels is a class-imbalanced problem, but a football game predictor in which 0.51 of examples label one team winning and 0.49 label the other team winning is not a class-imbalanced problem.
Clipping	A technique for handling outliers. Specifically, reducing feature values that are greater than a set maximum value down to that maximum value. Also, increasing feature values that are less than a specific minimum value up to that minimum value.
Confusion matrix	An NxN table that summarizes how successful the classification model predictions were; that is, the correlation between the label and the model classification. One axis of a confusion matrix is the label that the model predicted, and the other axis is the actual label.
Cross-validation	A mechanism for estimating how well a model generalizes to new data by testing the model against one or more non-overlapping data subsets withheld from the training set.
Data analysis	Obtaining an understanding of data by considering samples, measurement, and visualization. Data analysis can be particularly useful when a dataset is first received, before one builds the first model. It is also crucial in understanding experiments and debugging problems with the system.
Data augmentation	Artificially boosting the range and number of training examples by transforming existing examples to create additional examples. For example, suppose images are one of your features, but your dataset doesn't contain enough image examples for the model to learn useful associations. Ideally, you'd add enough labeled images to your dataset to enable your model to train properly. If that's not possible, data augmentation can rotate, stretch, and reflect each image to produce many variants of the original picture, possibly yielding enough labeled data to enable excellent training.
Data set or dataset	A collection of examples.
Decision boundary	The separator between classes learned by a model in a binary class or multi-class classification problems.
Decision threshold	Synonym for classification threshold.
Decision tree	A model represented as a sequence of branching statements.
Discrete feature	A feature with a finite set of possible values. For example, a feature whose values may only be animal, vegetable, or mineral is a discrete (or categorical) feature. Contrast with continuous feature.
Discriminative model	A model that predicts labels from a set of one or more features. More formally, discriminative models define the conditional probability of an output given the features and weights.

Downsampling	Overloaded term that can mean either of the following: • Reducing the amount of information in a feature in order to train a model more efficiently. • Training on a disproportionately low percentage of over-represented class examples in order to improve model training on under-represented classes.
Dynamic model	A model that is trained online in a continuously updating fashion. That is, data is continuously entering the model.
Example	One row of a dataset. An example contains one or more features and possibly a label. See also labeled example and unlabeled example.
False negative (FN)	An example in which the model mistakenly predicted the negative class. For example, the model inferred that a particular email message was not spam (the negative class), but that email message actually was spam.
False positive (FP)	An example in which the model mistakenly predicted the positive class. For example, the model inferred that a particular email message was spam (the positive class), but that email message was actually not spam.
False positive rate (FPR)	The x-axis in an ROC curve. The false positive rate is defined as follows: False positive rate = false positives / (false positives + true negatives)
Feature	An input variable used in making predictions.
Feature engineering	The process of determining which features might be useful in training a model, and then converting raw data from log files and other sources into said features. Feature engineering is sometimes called feature extraction.
Feature extraction	Overloaded term having either of the following definitions: • Retrieving intermediate feature representations calculated by an unsupervised or pre-trained model for use in another model as input. • Synonym for feature engineering.
Feature set	The group of features your machine learning model trains on. For example, postal code, property size, and property condition might comprise a simple feature set for a model that predicts housing prices.
Generalization	Refers to your model's ability to make correct predictions on new, previously unseen data as opposed to the data used to train the model.
Ground truth	The correct answer. Reality. Since reality is often subjective, expert raters typically are the proxy for ground truth.
Heuristic	A quick solution to a problem, which may or may not be the best solution.
Imbalanced dataset	Synonym for class-imbalanced dataset.
Independently and identically distributed (i.i.d)	Data drawn from a distribution that doesn't change, and where each value drawn doesn't depend on values that have been drawn previously. An i.i.d. is the ideal gas of machine learning—a useful mathematical construct but almost never exactly found in the real world. For example, the distribution of visitors to a web page may be i.i.d. over a brief window of time; that is, the distribution doesn't change during that brief window and one person's visit is generally independent of another's visit. However, if you expand that window of time, seasonal differences in the web page's visitors may appear.
Interference	In machine learning, often refers to the process of making predictions by applying the trained model to unlabeled examples. In statistics, inference refers to the process of fitting the parameters of a distribution conditioned on some observed data.
Instance	Synonym for example.
Interpretability	The degree to which a model's predictions can be readily explained. Deep models are often non-interpretable; that is, a deep model's different layers can be hard to decipher. By contrast, linear regression models and wide models are typically far more interpretable.
J48	An open source Java implementation of the C4.5 algorithm
Label	In supervised learning, the "answer" or "result" portion of an example. Each example in a labeled dataset consists of one or more features and a label. For instance, in a housing dataset, the features might include the number of bedrooms, the number of bathrooms, and the age of the house, while the label might be the house's price. In a spam detection dataset, the features might include the subject line, the sender, and the email message itself, while the label would probably be either "spam" or "not spam."
Linear regression	A type of regression model that outputs a continuous value from a linear combination of input features.

Machine learning	A program or system that builds (trains) a predictive model from input data. The system uses the learned model to make useful predictions from new (never-before-seen) data drawn from the same distribution as the one used to train the model. Machine learning also refers to the field of study concerned with these programs or systems.
Majority class	The more common label in a class-imbalanced dataset. For example, given a dataset containing 99% non-spam labels and 1% spam labels, the non-spam labels are the majority class.
Matplotlib	An open-source Python 2D plotting library. matplotlib helps you visualize different aspects of machine learning.
Minority class	The less common label in a class-imbalanced dataset. For example, given a dataset containing 99% non-spam labels and 1% spam labels, the spam labels are the minority class.
ML	Abbreviation for machine learning.
Model training	The process of determining the best model.
Multi-class classification	Classification problems that distinguish among more than two classes. For example, there are approximately 128 species of maple trees, so a model that categorized maple tree species would be multi-class. Conversely, a model that divided emails into only two categories (spam and not spam) would be a binary classification model.
Multinomial classification	Synonym for multi-class classification.
Negative class	In binary classification, one class is termed positive and the other is termed negative. The positive class is the thing we're looking for and the negative class is the other possibility. For example, the negative class in a medical test might be "not tumor." The negative class in an email classifier might be "not spam." See also positive class.
Neural network	A model that, taking inspiration from the brain, is composed of layers (at least one of which is hidden) consisting of simple connected units or neurons followed by nonlinearities.
Node (decision tree)	A "test" on an attribute.
Noise	Broadly speaking, anything that obscures the signal in a dataset. Noise can be introduced into data in a variety of ways. For example: • Human raters make mistakes in labeling. • Humans and instruments mis-record or omit feature values.
Normalization	The process of converting an actual range of values into a standard range of values, typically -1 to +1 or 0 to 1. For example, suppose the natural range of a certain feature is 800 to 6,000. Through subtraction and division, you can normalize those values into the range -1 to +1. See also scaling.
Numerical data	Features represented as integers or real-valued numbers.
Outliers	Values distant from most other values. In machine learning, any of the following are outliers: • Weights with high absolute values. • Predicted values relatively far away from the actual values. • Input data whose values are more than roughly 3 standard deviations from the mean. Outliers often cause problems in model training. Clipping is one way of managing outliers.
Overfitting	Creating a model that matches the training data so closely that the model fails to make correct predictions on new data.
Parameter	A variable of a model that the ML system trains on its own.
Performance	Overloaded term with the following meanings: • The traditional meaning within software engineering. Namely: How fast (or efficiently) does this piece of software run? • The meaning within ML. Here, performance answers the following question: How correct is this model? That is, how good are the model's predictions?
Positive class	In binary classification, the two possible classes are labeled as positive and negative. The positive outcome is the thing we're testing for. (Admittedly, we're simultaneously testing for both outcomes, but play along.) For example, the positive class in a medical test might be "tumor." The positive class in an email classifier might be "spam." Contrast with negative class.

Precision	A metric for classification models. Precision identifies the frequency with which a model was correct when predicting the positive class. That is: Precision = true positives / (true positives + false positives)
Prediction	A model's output when provided with an input example.
Pre-trained model	Models or model components that have been already been trained.
Proxy labels	Data used to approximate labels not directly available in a dataset. For example, suppose you want “is it raining?” to be a Boolean label for your dataset, but the dataset doesn't contain rain data. If photographs are available, you might establish pictures of people carrying umbrellas as a proxy label for “is it raining”? However, proxy labels may distort results. For example, in some places, it may be more common to carry umbrellas to protect against sun than the rain.
Rater	A human who provides labels in examples. Sometimes called an "annotator."
Recall	A metric for classification models that answers the following question: “Out of all the possible positive labels, how many did the model correctly identify?” That is: Recall = true positives / (true positives + false negatives)
Regression model	A type of model that outputs continuous (typically, floating-point) values. Compare with classification models, which output discrete values, such as "day lily" or "tiger lily."
Reinforcement learning	A machine learning approach to maximize an ultimate reward through feedback (rewards and punishments) after a sequence of actions. For example, the ultimate reward of most games is victory. Reinforcement learning systems can become expert at playing complex games by evaluating sequences of previous game moves that ultimately led to wins and sequences that ultimately led to losses.
Representation	The process of mapping data to useful features.
ROC curve	ROC = receiver operating characteristic A curve of true positive rate vs. false positive rate at different classification thresholds.
Scaling	A commonly used practice in feature engineering to tame a feature's range of values to match the range of other features in the dataset. For example, suppose that you want all floating-point features in the dataset to have a range of 0 to 1. Given a particular feature's range of 0 to 500, you could scale that feature by dividing each value by 500. See also normalization.
Scikit-learn	A popular open-source ML platform. See www.scikit-learn.org.
Scoring	The part of a recommendation system that provides a value or ranking for each item produced by the candidate generation phase.
Semi-supervised learning	Training a model on data where some of the training examples have labels but others don’t. One technique for semi-supervised learning is to infer labels for the unlabeled examples, and then to train on the inferred labels to create a new model. Semi-supervised learning can be useful if labels are expensive to obtain but unlabeled examples are plentiful.
Sequence model	A model whose inputs have a sequential dependence. For example, predicting the next video watched from a sequence of previously watched videos.
Serving	A synonym for inferring.
Static model	A model that is trained offline.
Stationarity	A property of data in a dataset, in which the data distribution stays constant across one or more dimensions. Most commonly, that dimension is time, meaning that data exhibiting stationarity doesn't change over time. For example, data that exhibits stationarity doesn't change from September to December.
Supervised machine learning	Training a model from input data and its corresponding labels. Supervised machine learning is analogous to a student learning a subject by studying a set of questions and their corresponding answers. After mastering the mapping between questions and answers, the student can then provide answers to new (never-before-seen) questions on the same topic. Compare with unsupervised machine learning.
Target	Synonym for label.
Training	The process of determining the ideal parameters comprising a model.
Training set	The subset of the dataset used to train a model.

	Contrast with validation set and test set.
True negative (TN)	An example in which the model correctly predicted the negative class. For example, the model inferred that a particular email message was not spam, and that email message really was not spam.
True positive (TP)	An example in which the model correctly predicted the positive class. For example, the model inferred that a particular email message was spam, and that email message really was spam.
True positive rate (TPR)	Synonym for recall. That is: True positive rate = true positives / (true positives + false negatives) True positive rate is the y-axis in an ROC curve.
Underfitting	Producing a model with poor predictive ability because the model hasn't captured the complexity of the training data. Many problems can cause underfitting, including: • Training on the wrong set of features. • Training for too few epochs or at too low a learning rate. • Training with too high a regularization rate. • Providing too few hidden layers in a deep neural network.
Unlabeled example	An example that contains features but no label. Unlabeled examples are the input to inference. In semi-supervised and unsupervised learning, unlabeled examples are used during training.
Unsupervised machine learning	Training a model to find patterns in a dataset, typically an unlabeled dataset. The most common use of unsupervised machine learning is to cluster data into groups of similar examples. For example, an unsupervised machine learning algorithm can cluster songs together based on various properties of the music. The resulting clusters can become an input to other machine learning algorithms (for example, to a music recommendation service). Clustering can be helpful in domains where true labels are hard to obtain. For example, in domains such as anti-abuse and fraud, clusters can help humans better understand the data. Another example of unsupervised machine learning is principal component analysis (PCA). For example, applying PCA on a dataset containing the contents of millions of shopping carts might reveal that shopping carts containing lemons frequently also contain antacids. Compare with supervised machine learning.
Validation	A process used, as part of training, to evaluate the quality of a machine learning model using the validation set. Because the validation set is disjoint from the training set, validation helps ensure that the model's performance generalizes beyond the training set. Contrast with test set.
Validation set	A subset of the dataset—disjoint from the training set—used in validation. Contrast with training set and test set.
Weka	A collection of machine learning algorithms for data mining tasks. It contains tools for data preparation, classification, regression, clustering, association rules mining, and visualization.

Revision history

Table 10. **Document revision history**

Date	Version	Changes
02-Feb-2023	1	Initial release

Contents

List of tables

List of figures